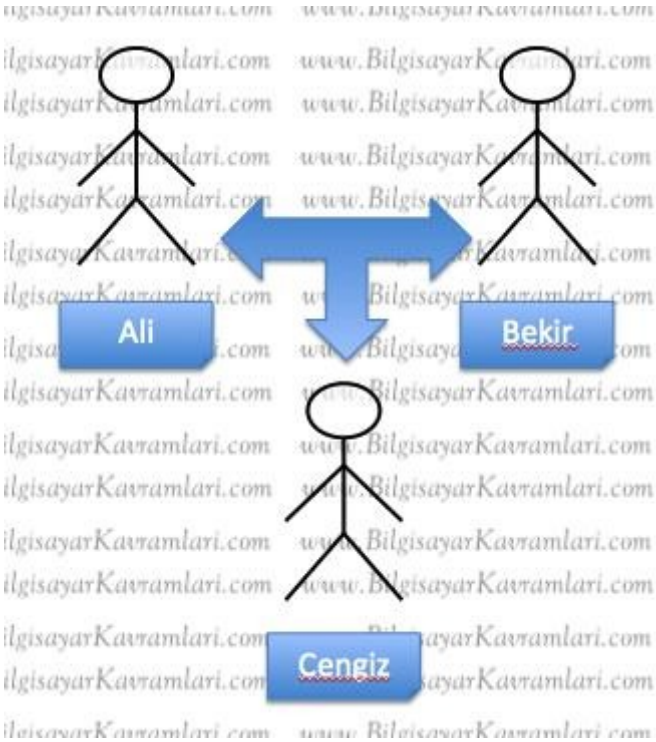
İçindekiler

BİLGİSAYAR STANDARTLARI

SORU-1: Security Protocol Notation (Güvenlik Teşrifat Yazımı) hakkında bilgi veriniz.

Bu yazının amacı, bilgisayar güvenliğinde kullanılan güvenlik teşrifat (protokol) yazımını (notation) açıklamaktır. Pek çok algoritmanın / protokolün (teşrifat) çalışmasını anlatırken standart bir gösterime gidilmiş ve bazı kurallar konulmuştur. Bu kurallara göre algoritmaları yazmak ve okumak daha kolay hale gelir.

Bu kurallara göre, öncelikle iletişim kuran iki taraf kabulu yapılır. Bu taraflar genelde alfabenin ilk iki harfi olan A ve B harfleri ile gösterilir. Ayrıca güvenlik ile ilgili kitaplarda okumayı kolaylaştırmak için Alice ve Bob isimleri kullanılmaktadır. Genelde bu iki kişinin iletişimine saldıran üçünü kişi ise C harfi ile başlayan Charlie olarak okunmaktadır. Bu yazı kapsamında Türkçe içerik sunulduğu için bu harflere karşılık Türkçedeki Ali, Bekir ve Cengiz isimleri kullanılacaktır. Kısacası temel bir iletişim yapısı aşğıdaki şekilde resmedilebilir.



Yukarıdaki şekilde görüldüğü üzere, Ali ve Bekir arasındaki iletişime Cengiz bir şekilde dahil olabilmektedir. Bu saldırının çok sayıda çeşidi olup detayları bu yazının konusu dışındadır. Bu yazı kapsamında bir protokol anlatılırken, Ali, Bekir ve Cengiz'in yapabileceklerini göstermenin kuralları açıklanacaktır.

Kullanılan kısaltmalar:

- A : Ali (mesaj gönderen taraflardan birisi)
- B : Bekir (mesaj gönderen diğer taraf veya çoğu durumda mesajı alan taraf)
- C : Cengiz (iletişime saldıran taraf)
- K : Anahtar (ingilizcedeki Key kelimesinden gelir)

- Ka : Ali'nin anahtarı (yanındaki harf sahibini gösterir)
- PK : Umumi anahtar (ingilizcedeki Public Key kelimelerinden gelir)
- PKa : Ali'nin umumi anahtarı (bazı kaynaklarda açık anahtar olarak da geçer)
- T : Zaman bilgisi (ingilizcedeki Timestamp kelimesinden gelir ve içinde zaman bilgisi taşıyan bir mesajdır)
- Na : Nonce (yanındaki harf kimin nonce bilgisi olduğunu anlatır, örnekte Ali'nin nonce bilgisi)
- X : Açık mesaj (plain text)

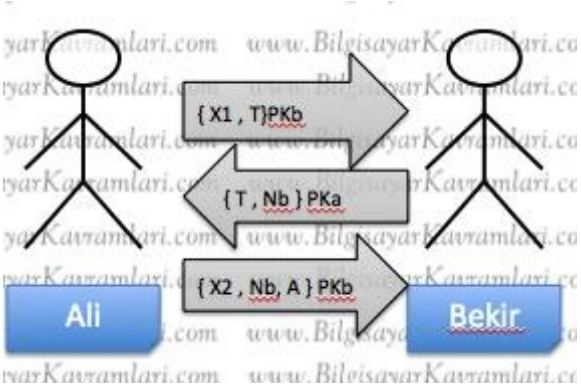
Yukarıdaki kısaltmaları kullanarak örneğin aşağıdaki teşrifatı (protocol) okuyalım:

1. A–> B : { X1 , T}PKb
2. B–> A : { T , Nb } PKa
3. A–> B : { X2 , Nb, A } PKb

Yukarıdaki gösterimde, 1. adımda, Ali, Bekir'e 1 numaralı açık mesajı göndermiş, gönderirken bu mesajın yanına bir zaman bilgisi ekleyerek Bekir'in umumi anahtarı ile şifrelemiştir.

2. adımda, Bekir, Ali'ye cevap olarak anlık bir zaman bilgisi oluşturmuş ve yanına kendi oluşturduğu nonce bilgisini eklemiş bu bilgileri Ali'nin umumi anahtarı ile şifrelemiştir.

Son adımda ise Ali, Bekir'e mesajın ikinci kısmını ve Bekir'in nonce bilgisini, kendisini tanıtan bir A bilgisi ile birlikte yollamıştır.



Şekilde gösterilen mesajlaşma aslında teşrifat yazımında kullanılan gösterimin görsel halidir.

Yukarıdaki örnekte, örneğin 3. adımda bulunan Nb bilgisinin, aslında 2. adımda alınan ve Bekir'in Ali'ye yolladığı nonce bilgisi olduğunu biliyoruz, çünkü nonce bilgisi ilgili kişi tarafından üretilebilir. Ali, Nb üretemeyeceğine göre, 3. adımda yollanan Nb'nin alınmasının tek yolu 2. adımda ulaşan Nb'nin bu adımda geri yollanmasıdır. Bu tip yollamaların sebebi genelde mesajın yolda bir saldırı sonucu bozulup bozulmadığını anlamaktır.

SORU-2: ICMP (Internet Kontrol Mesajı Protokolü) hakkında bilgi veriniz.

İngilizce, Internet Control Message Protocol kelimelerinin baş harflerinden oluşan kısaltmadır. Türkçe olarak İnternet Tespit Mesajı Teşrifatı olarak çevrilebilir.

Genel olarak işletim sistemleri tarafından, ağda bulunan cihazların durumunu tespit amaçlı kullanılan bir teşrifattır (protocol). Örneğin bir cihaza erişilip erişilemediğini tespit için gönderilen mesaj tipidir.

ICMP mesajları, birer IP paketi halinde yollanmaktadır. ICMP hem UDP hem de TCP üzerinde çalışabilmektedir ancak çalıştığı protokole göre farklılık gösterebilir. Örneğin UDP paketlerine itibar etmek doğru olmaz çünkü paket kaybı olabilir.

ICMP paketleri ayrıca üzerinde çalıştıkları IP sürümüne göre (IP Version) ismlendirilmektedirler. Örneğin IPv4 için olan paketlere ICMPv4, IPv6 için olanlara ise ICMPv6 ismi verilmektedir.

ICMP paketlerinin 8 byte uzunluğunda bir başlığı (header) ve bunu takip eden ve değişken boyutta veri kısmı bulunur. Klasik bir windows ICMP paketi 32 byte uzunluğundaykey, klasik bir UNIX / Linux paketi ise 64 byte uzunluğundadır. Bu durumda windows için veri uzunluğu 24 byte, linux için ise 56 byte olmaktadır.

ICMP paketinin başlığında ilk byte, ICMP tipini belirtir. İkinci byte ICMP kodunu, üçüncü ve dördüncü bytelar ise paketin tamamının, toplam kontrolünü (check sum) belirtir. Başlık boyutunun 8 byte olduğunu belirtimiştik, geri kalan 4 byte ise ICMP tip ve koduna göre değişiklik göstermektedir. Bu durumda ICMP paketi aşağıdaki yapıda olacaktır:

Bitler	0–7	8–15	16–23	24–31
0	Tip	Kod	Toplam Kontrolü	
32	Başlığın tipine ve koduna bağlı olarak devamı			

Yukarıdaki tiplerin değerine örnek olması açısından aşağıdaki tablodan istifade edilebilir:

Tip	Açıklama
0	Eko yanıt-ping yanıtı(Echo Reply)
3	Hedefe Erişilemedi(Destnation Not Reachable)
4	Kaynak Kapatmak(Source Quench)
5	Yeniden Yönlendirme(Redirection Required)
8	Eko yanıt-ping isteği(Echo Request)
9	Yönlendirici tanıtımı
10	Yönlendirici istemi
11	Zaman aşımı–traceroute kullanır(Time to Live Exceeded)

12	Parametre Problemi(Parameter Problem)
13	Timestamp İstemi(Timestamp Request)
14	Timestamp Yanıtı(Timestamp Reply)
15	Bilgi İstemi(Information Request)
16	Bilgi Yanıtı(Information Reply)
17	Addres Maskesi istemi(Address Mask Request)
18	Addres Maskesi yanıtı(Address Mask Reply)

Yukarıdaki her tip için ayrıca kodlar bulunmaktadır.

Örneğin 5 numaralı tip olan yeniden yönlendirme tipinin (redirection required) alt kodları olarak aşağıdaki tabloda yer alan değerlerden birisi atanabilir:

Kod	Açıklama
0	Ağ için yönlendir (Redirect Datagram for the Network)
1	Sahibi için yönlendir (Redirect Datagram for the Host)
2	Servis tipine göre ve ağa göre yönlendir (Redirect Datagram for the TOS & network)
3	Servis tipine ve sahibine göre yönlendir (Redirect Datagram for the TOS & host)

Ayrıca diğer tiplerinde benzer şekilde alt kodları bulunmaktadır. ICMP paketi, ayrıca internet standartlarını belirleyen kurum olan IETF (internet engineerint task force) tarafından yayınlanan ve standartların yayınlandığı RFC dokümanlarında (yorum için talep, request for comment) 792 numaralı yayında geçmektedir. İlgili dokümana aşağıdaki bağlantıdan erişilebilir:

http://tools.ietf.org/html/rfc792

SORU-3: ASCII Tablosu hakkında bilgi veriniz.

Bilgisayar bilimlerinde, farklı sembolleri sayısal olarak göstermek için kullanılan standardın ismidir. American Standard Code for Information Interchange kelimelerinin baş harflerinden oluşur. Tablonun genişletilmiş şekilleri olmasına karşılık temel tablo 7 bit için 2^7 ihtimal kodlar ve bu da 128 farklı sembole karşılık gelir:

```
 Dec Hex Char                        Dec Hex Char   Dec Hex Char   Dec Hex
Char
 ----------                          ------------  ------------
------------
  0   0  NUL  (null)                  32  20         64  40  @      96  60
`
  1   1  SOH  (start of heading)      33  21  !      65  41  A      97  61
a
  2   2  STX  (start of text)         34  22  "      66  42  B      98  62
b
  3   3  ETX  (end of text)           35  23  #      67  43  C      99  63
c
```

4	4	EOT	(end of transmission)	36	24	$	68	44	D	100	64
d											
5	5	ENQ	(enquiry)	37	25	%	69	45	E	101	65
e											
6	6	ACK	(acknowledge)	38	26	&	70	46	F	102	66
f											
7	7	BEL	(bell)	39	27	'	71	47	G	103	67
g											
8	8	BS	(backspace)	40	28	(	72	48	H	104	68
h											
9	9	TAB	(horizontal tab)	41	29	)	73	49	I	105	69
i											
10	A	LF	(NL line feed, new line)	42	2A	*	74	4A	J	106	6A
j											
11	B	VT	(vertical tab)	43	2B	+	75	4B	K	107	6B
k											
12	C	FF	(NP form feed, new page)	44	2C	,	76	4C	L	108	6C
l											
13	D	CR	(carriage return)	45	2D	-	77	4D	M	109	6D
m											
14	E	SO	(shift out)	46	2E	.	78	4E	N	110	6E
n											
15	F	SI	(shift in)	47	2F	/	79	4F	O	111	6F
o											
16	10	DLE	(data link escape)	48	30	0	80	50	P	112	70
p											
17	11	DC1	(device control 1)	49	31	1	81	51	Q	113	71
q											
18	12	DC2	(device control 2)	50	32	2	82	52	R	114	72
r											
19	13	DC3	(device control 3)	51	33	3	83	53	S	115	73
s											
20	14	DC4	(device control 4)	52	34	4	84	54	T	116	74
t											
21	15	NAK	(negative acknowledge)	53	35	5	85	55	U	117	75
u											
22	16	SYN	(synchronous idle)	54	36	6	86	56	V	118	76
v											
23	17	ETB	(end of trans. block)	55	37	7	87	57	W	119	77
w											
24	18	CAN	(cancel)	56	38	8	88	58	X	120	78
x											
25	19	EM	(end of medium)	57	39	9	89	59	Y	121	79
y											
26	1A	SUB	(substitute)	58	3A	:	90	5A	Z	122	7A
z											
27	1B	ESC	(escape)	59	3B	;	91	5B	[	123	7B
{											
28	1C	FS	(file separator)	60	3C	<	92	5C	\	124	7C
|											
29	1D	GS	(group separator)	61	3D	=	93	5D	]	125	7D
}											
30	1E	RS	(record separator)	62	3E	>	94	5E	^	126	7E
~											
31	1F	US	(unit separator)	63	3F	?	95	5F	_	127	7F
DEL											

SORU-4: Ve/Veya bağlacı normal şeklleri (Conjunction / Disjunction Normal Form) hakkında bilgi veriniz.

Bu gösterim, bool cebirinde (boolean algebra) kullanılan ve kaziyeleri (önerme, proposition) ve (and) bağlacı ile bağlamanın özel bir şeklidir. Kısaca CNF (conjuction normal form) olarak ifade edilir. Diğer bir normal şekil olan Chomsky Normal Form (CNF) ile ilgili bilgi arıyorsanız buradan ulaşabilirsiniz.

Bu özel şeklin taşıdığı kuralları aşağıdaki şekilde sıralayabiliriz:

Ters (not) işlemi, sadece önermelerin lafzi gösterimlerin (literal) başında bulunacaktır. Bir kaziyenin (önerme, propositon) başında bulunmayacaktır.

Örneğin ¬A şeklindeki tek lafzi gösterim üzerinde olan ters işlemi kabul edilirken ¬(A V B) şekli kabul edilemez.

Ve (and) işlemi, iki operand almaktadır. Yani a ve b şeklindeki bir işlemde a ve b adında iki lafzi gösterim (literal) bulunmaktadır. Bu işlemin herhangi bir operandı, basit bir lafzi gösterim (literal) veya disjunction (veya operatörü (or opeartor) ile oluşturulmuş bir) kaziye (önerme, proposition) olmalıdır.

Örneğin aşağıdaki gösterimler bu kurala uymaktadır:

A Λ B

(A V B) Λ (C V D)

A Λ (C V D V F)

Ancak aşağıdaki gösterimler bu kurala uymaz

A Λ (B Λ C)

A Λ (B V C Λ D)

(A Λ B) Λ C

Ancak yukarıdaki gösterimler CNF (conjuction normal form) yani ve bağlacı normal şekline çevrilebilir. Bu çevirimler aşağıda gösterilmiştir (yukarıdaki her satır için sırasıyla)

A Λ B Λ C

A Λ (B V C) Λ (B V D)

A Λ B Λ C

Görüldüğü üzere yukarıdaki yeni hallerinde, verilen gösterimler ve bağlacı üzerinde basitleştirilmiştir.

Disjunction Normal Form (DNF, Veya bağlacı normal şekli) ise yukarıdaki ve bağlacı için söylenen her şeyin, veya bağlacı için geçerli olması durumudur.

Kısaca bu şekilde kabul edilemeyecek gösterimleri ve çevrilmiş şekillerini aşağıda verelim:

DNF şeklinde olmayanlar	DNF karşılığı
(A V B) Λ C	(A Λ C) V (B Λ C)
A Λ (B V C Λ D)	(A Λ B) V (A Λ (C Λ D))
¬(A Λ B)	¬A V ¬B

İki gösterimin doğruluğunun tespiti. Yani verilen bir boole cebiri diziliminin gerçekten CNF veya DNF olduğunun sınanması, veya verilen bir denklemin sonucunun doğru (true) veya yanlış (false) olarak bulunabilmesi, bu denklemdeki lafzi terim (literal) sayısına bağlıdır. Örneğin 3CNF bir denklemin, yani sadece 3 terimin conjunction (ve bağlacı ile bağlanmasından) oluşan bir denklemin karmaşıklığı NP-Complete olurken, bu terim sayısı 2′ye indirildiğinde yani 2CNF bir denklem söz konusu olduğunda, karmaşıklık P zamanda (polinom zamanda) çözülebilmektedir.
Daha detaylı bilgi için ikili tatmin problemleri (boolean satisfaction problem) başlıklı yazıyı okuyabilirsiniz.

SORU-5: JML (Java Modelleme Dili) hakkında bilgi veriniz.

JML ingilizce Java modelling language kelimelerinin baş harflerinden oluşan bir kısaltmadır. Basitçe bir java kaynak koduna eklenen ilave satırlar ile progam doğruluğunu (program correctness) sağlamayı amaçlar (program verification).

İlave olarak eklenecek satırlar java kodunun içerisine yorum satırı gibi ilave edilir. Normal java yorum satırlarından tek farkı ilave olarak konulan @ işaretidir.

Örneğin

```
//@ örnek jml satırı
```

veya

```
/*@ örnek
çoklu jml
satırı @*/
```

şeklinde java koduna ilave edilebilirler.

Aslında bütün jml komutlarını basit üç grupta toplamak mümkündür:

1. Önkoşullar (preconditions)
2. Sonkoşullar (postconditions)
3. Değişmezler (invariants)

Yukarıdaki bu üç grubtan ilk ikisi hoare mantığındaki ön ve son koşullara benzetilebilir. Üçüncü grup ise döngülere özgü kullanılan bir koşuldur. Bu gruplardaki komutları ve kullanımlarını aşağıdaki örneklerden inceleyelim:

```
static void arraycopy (int[] src, int srcPos,int[] dest, int destPos,int
len);
```

Örnek olarak yukarıdaki dizi (array) kopyalama fonksiyonunu ele alalım. Burada amaç, src dizisinden dest dizisine derin kopyalama yapmaktır. (deep copy). Ancak bu satırda yaşanabilecek bazı programlar aşağıdaki şekildedir:

- src dizisi null olabilir
- dest dizisi null olabilir
- src dizisinin izin verilen eleman sayısının dışına programcı tarafından çıkılabilir (örneğin 10 elemanlı bir dizinin 10. elemanına erişilebilir (java terminolojisi olarak arrayindexoutofboundsexception olabilir))
- dest dizisinin izin verilen eleman sayısının dışına programcı tarafından çıkılabilir.

Yukarıdaki bu olası problemleri çözmek için fonksiyonu yazarken başına ilave olarak bu requires komutu ile ön koşullarımızı ekliyoruz.

```
/*@ requires src != null && dest != null &&
0 <= srcPos && srcPos + len < src.length &&
0 <= destPos && srcPos + len < dest.length;
@*/
static void arraycopy (int[] src, int srcPos,int[] dest, int destPos,int
len);
```

Yukarıdaki yeni kodda ilave olarak src ve dest'in null olamayacakları ve kopyalama sırasında kullanılan indis değişkeni olan srcPos'un 0 ile src.length ve dest.length arasında olacağı belirtilmiştir.

Yukarıdaki ön koşula ilave olarak diğer bir isteğimiz fonksiyon çalıştıktan sonra src ile dest dizilerinin elemanlarının birebir aynı olmalarıdır. Bu son koşulu (postcondition) kontrol etmek için ilave bir jml satırı daha eklememiz gerekir:

```
/*@ requires src != null && dest != null &&
0 <= srcPos && srcPos + len < src.length &&
0 <= destPos && srcPos + len < dest.length;
ensures (forall int i; 0 <= i && i < len;dest[dstPos+i] == src[srcPos+i] )
@*/
static void arraycopy (int[] src, int srcPos,int[] dest, int destPos,int len);
```

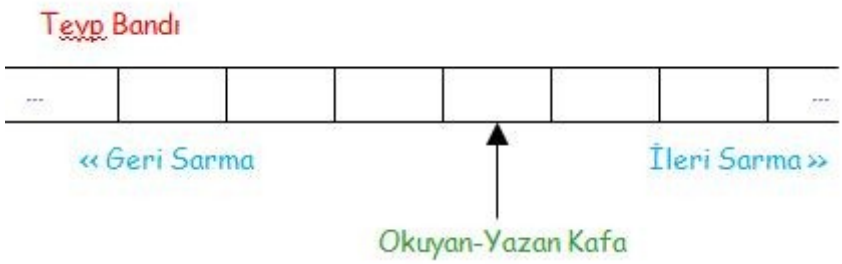
Yukarıdaki yeni jml satırı olan ensures ile son koşul kontrolü yapılıyor. Burada yeni kullanılan forall döngüsü sayesinde 0′dan len değişkenine kadar olan değerler için (döngü değişkeni i olarak) bütün dizi içeriği karşılaştırılıyor ve ancak aynıysa son koşul sağlanmış oluyor.

SORU-6: Turing Makinesi (Turing Machine) hakkında bilgi veriniz.

Bilgisayar bilimlerinin önemli bir kısmını oluşturan otomatlar (Automata) ve Algoritma Analizi (Algorithm analysis) çalıştırmalarının altındaki dil bilimin en temel taşlarından birisidir.1936 yılında Alan Turing tarafından ortaya atılan makine tasarımı günümüzde pekçok teori ve standardın belirlenmesinde önemli rol oynar.

Turing Makinesinin Tanımı

Basitçe bir kafadan (head) ve bir de teyp bandından (tape) oluşan bir makinedir.



Makinede yapılabilecek işlemler

- Yazmak
- Okumak
- Bandı ileri sarmak
- Bandı geri sarmak

şeklinde sıralanabilir.

Chomsky hiyerarşisi ve Turing Makinesi

Bütün teori bu basit dört işlem üzerine kurulmuştur ve sadece yukarıdaki bu işlemleri kullanarak bir işin yapılıp yapılamayacağı veya bir dilin bu basit 4 işleme indirgenip indirgenemeyeceğine göre diller ve işlemler tasnif edilmiştir.

Bu sınıflandırma yukarıdaki venn şeması ile gösterilmiştir. Aynı zamanda chomsky hiyerarşisi (chomsky hierarchy) için 1. seviye (type-1) olan ve Turing makinesi ile kabul edilebilen diller bütün tip-2 ve tip-3 dilleri yani içerk bağımsız dilleri ve düzenli dilleri kapsamaktadır. Ayrıca ilave olarak içerik bağımsız dillerin işleyemediği (üretemediği veya parçalayamadığı (parse)) anbncn şeklindeki kelimeleri de işleyebilmektedir. Düzenli ifadelerin işleyememesi konusunda bilgi için düzenli ifadelerde pompalama savı (pumping lemma in regular expressions) ve içerik bağımsız dillerin işlemeyemesi için de içerik bağımsız dillerde pompalama savı (pumping lemma for CFG) başlıklı yazıları okuyabilirsiniz.

Turing Makinesinin Akademik Tanımı

Turing makineleri literatürde akademik olarak aşağıdaki şekilde tanımlanır:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Diamond, F)$$

Burada M ile gösterilen makinenin parçaları aşağıda listelenmiştir:

Q sembolü sonlu sayıdaki durumların kümesidir. Yani makinenin işleme sırasında aldığı durumardır.

Γ sembolü dilde bulunan bütün harfleri içeren alfabeyi gösterir. Örneğin ikilik tabandaki sayılar ile işlem yapılıyorsa {0,1} şeklinde kabul edilir.

Σ sembolü ile makineye verilecek girdiler (input) kümesi gösterilir. Girdi kümesi dildeki harfler dışında bir sembol taşıyamayacağı için $\Sigma \subseteq \Gamma$ demek doğru olur.

δ sembolü dilde bulunan ve makinenin çalışması sırasında kullanacağı geçişleri (transitions) tutmaktadır.

◊ sembolü teyp bandı üzerindeki boşlukları ifade etmektedir. Yani teyp üzerinde hiçbir bilgi yokken bu sembol okunur.

q_0 sembolü makinenin başlangıç durumunu (state) tutmaktadır ve dolayısıyla $q_0 \subseteq Q$ olmak zorundadır.

F sembolü makinenin bitiş durumunu (state) tutmaktadır ve yine $F \subseteq Q$ olmak zorundadır.

Örnek Turing Makinesi

Yukarıdaki sembolleri kullanarak örnek bir Turing makinesini aşağıdaki şekilde inşa edebiliriz.

Örneğin basit bir kelime olan a* düzenli ifadesini (regular expression) Turing makinesi ile gösterelim ve bize verilen aaa şeklindeki 3 a yı makinemizin kabul edip etmediğine bakalım.

Tanım itibariyle makinemizi aşağıdaki şekilde tanımlayalım:

$M = \{ \{q_0,q_1\} , \{ a \} , \{ a,x \} , \{ q_0\ a \rightarrow a\ R\ q_0 , q_0\ x \rightarrow x\ L\ q_1 \} , q_0 , x , q_1 \}$

Yukarıdaki bu makineyi yorumlayacak olursak:

Q değeri olarak $\{q_0,q_1\}$ verilmiştir. Yani makinemizin ik idurumu olacaktır.

Γ değeri olarak { a,x } verilmiştir. Yani makinemizdeki kullanılan semboller a ve x'ten ibarettir.

Σ değeri olara {a} verilmiştir. Yani makinemize sadece a girdisi kabul edilmektedir.

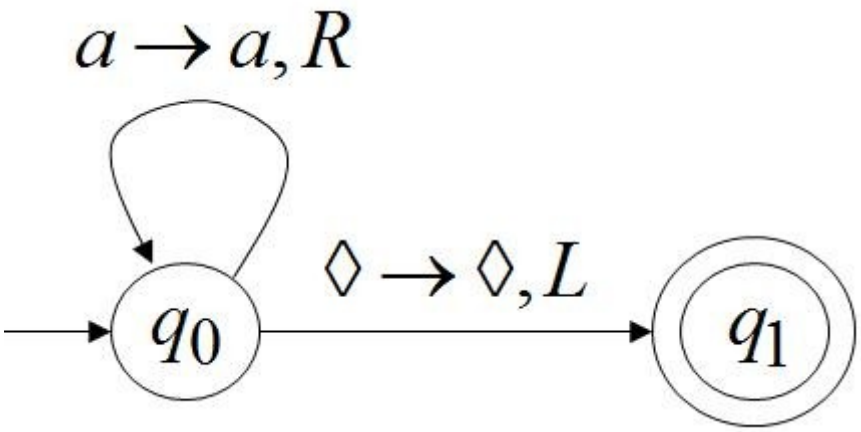
δ değeri olarak iki geçiş verilmiştir $\{ q_0\ a \rightarrow a\ R\ q_0 , q_0\ x \rightarrow x\ L\ q_1 \}$ buraadki R sağa sarma L ise sola sarmadır ve görüleceği üzere Q değerindeki durumlar arasındaki geçişleri tutmaktadır.

◊ değeri olarak x sembolü verilmiştir. Buradan x sembolünün aslında boş sembolü olduğu ve bantta hiçbir değer yokken okunan değer olduğu anlaşılmaktadır.

q_0 ile makinenin başlangıç durumundaki hali belirtilmiştir.

F değeri olarak q_1 değeri verilmiştir. Demek ki makinemiz q_1 durumuna geldiğinde bitmektedir (halt) ve bu duruma gelmesi halinde bu duruma kadar olan girdileri kabul etmiş olur.

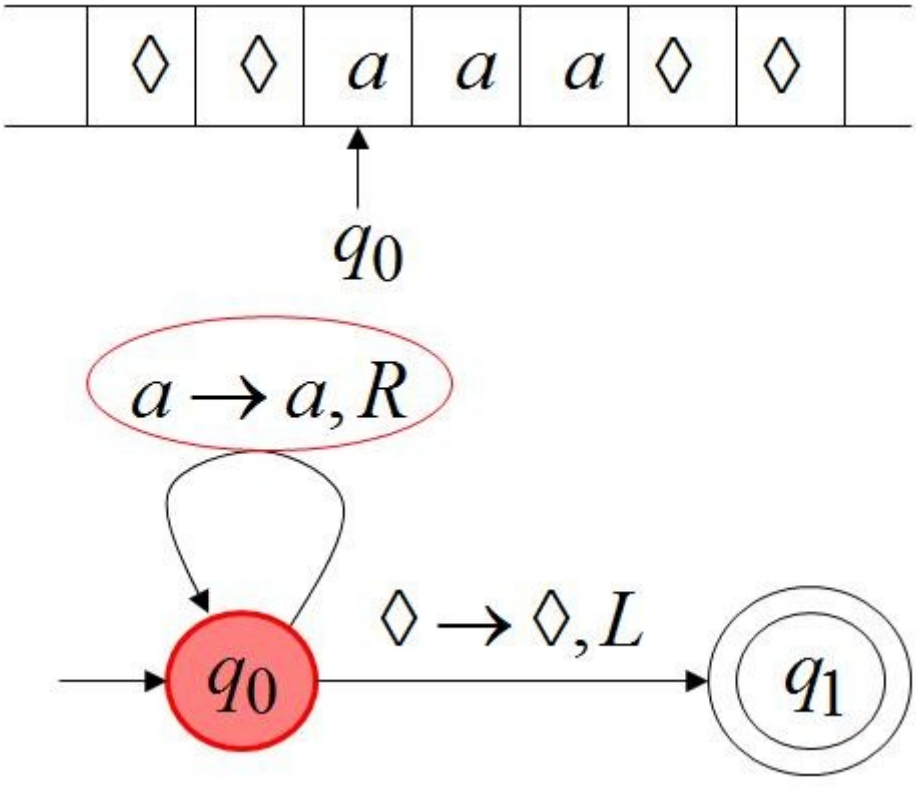
Yukarıdaki bu tanımı görsel olarak göstermek de mümkündür:



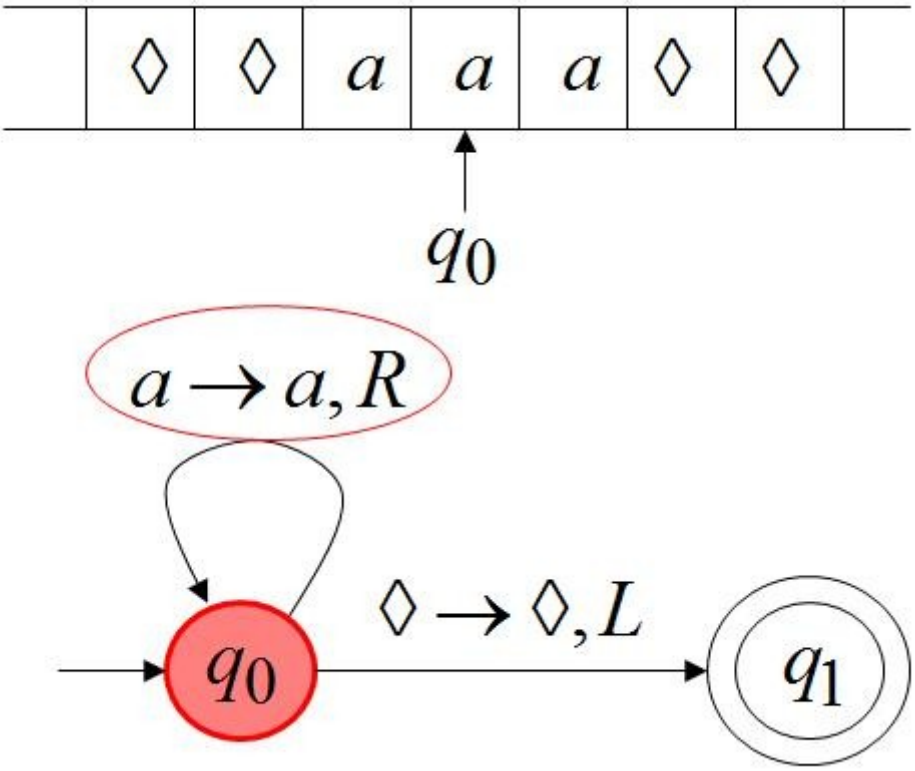
Yukarıdaki bu temsili resimde verilen turing makinesi çizilmiştir.

Makinemizin örnek çalışmasını ve bant durumunu adım adım inceleyelim.

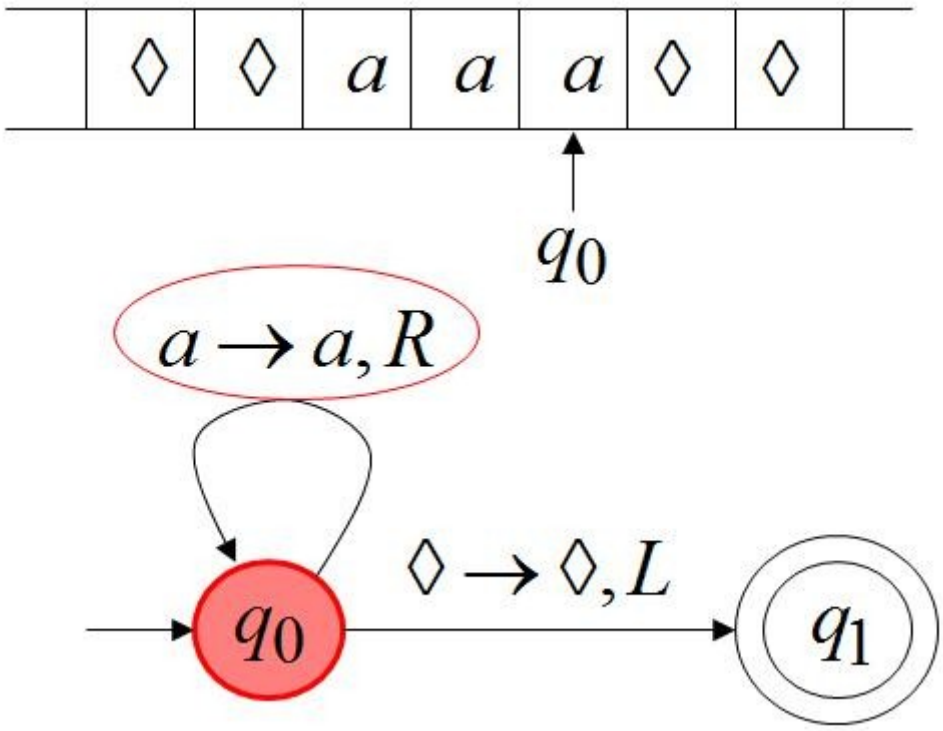
Birinci adımda bandımızda aaa (3 adet a) yazılı olduğunu kabul edelim ve makinemizin bu aaa değerini kabul edip etmeyeceğini adım adım görelim. Zaten istediğimiz de aaa değerini kabul eden bir makine yapabilmekti.



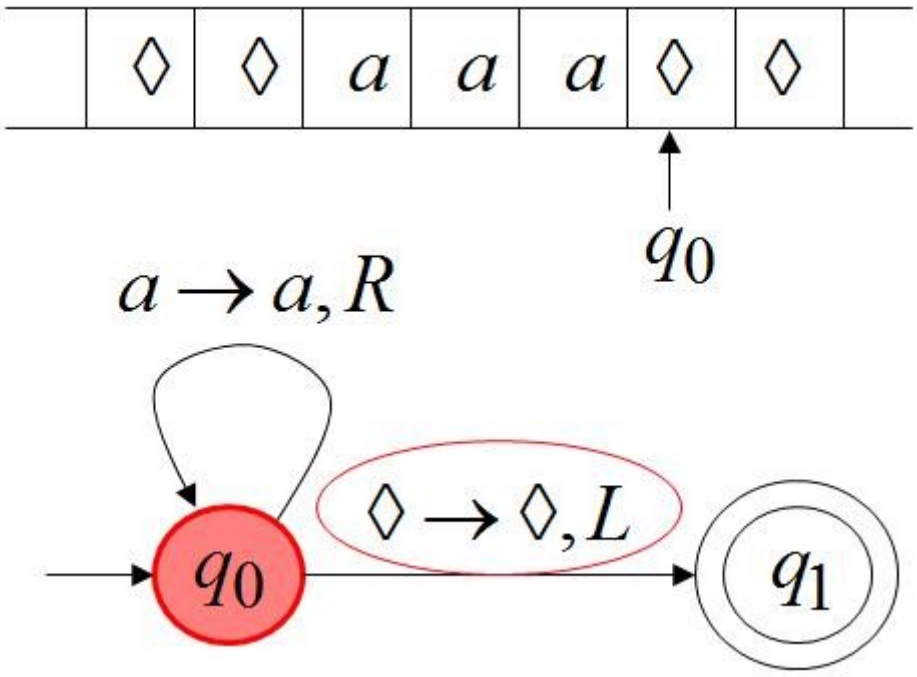
Yukarıdaki ilk durumda bant üzerinde beklenen ve kabul edilip edilmeyeceği merak edilen değerimiz bulunuyor. Makinemizin kafasının okuduğu değer a sembolü. Makinemizin geçiş tasarımına göre q_0 halinde başlıyoruz ve a geldiğinde teybi sağa sarıp yine q_0 durumunda kalmamız gerekiyor.



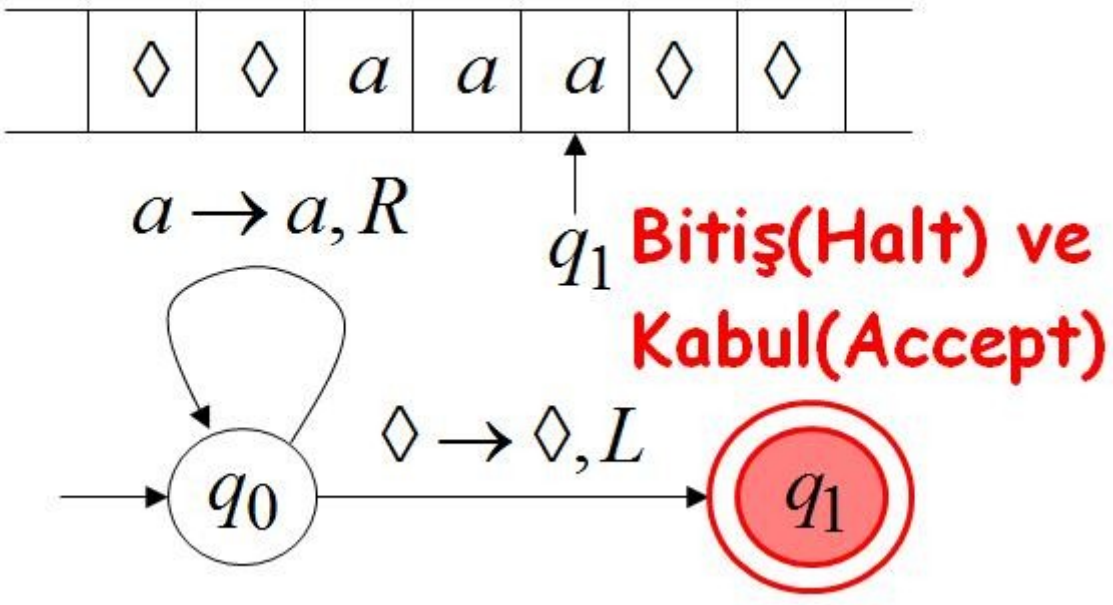
Yeni durumda kafamızın okuduğu değer banttaki 2. a harfi ve bu durumda yine q_0 durumundayken teybi sağa sarıp yine q_0 durumunda kalmamız tasarlanmış



3. durumda kafamızın okuduğu değer yine a sembolü olmakta ve daha önceki 2 duruma benzer şekilde q_0 durumundayken a sembolü okumanın sonucu olarak teybi sağa sarıp q_0 durumunda sabit kalıyoruz.



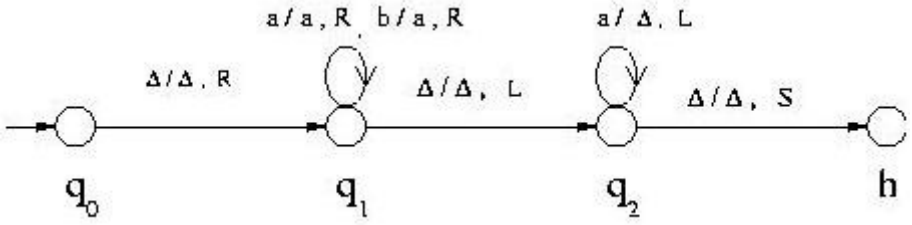
4. adımda teypten okuduğumuz değer boşluk sembolü x oluyor. Bu değer makinemizin tasarımında q_1 durumuna gitmemiz olarak tasarlanmış ve teybe sola sarma emri veriyoruz.



Makinenin son durumunda q_1 durumu makinenin kabul ve bitiş durumu olarak tasarlanmıştı (makinenin tasarımındaki F kümesi) dolayısıyla çalışmamız burada sonlanmış ve giriş olarak aaa girdisini kabul etmiş oluyoruz.

2. Örnek

Hasan Bey'in sorusu üzerine bir örnek makine daha ekleme ihtiyacı zuhur etti. Makinemiz {a,b} sembolleri için çalışsın ve ilk durum olarak bandın en solunda başlayarak bantta bulunan sembolleri silmek için tasarlansın. Bu tasarımı aşağıdaki temsili resimde görülen otomat ile yapabiliriz:



Görüldüğü üzere makinemizde 4 durum bulunuyor, bunlardan en sağda olan h durumu bitişi (halt) temsil ediyor. Şimdi bu makinenin bir misal olarak "aabb" yazılı bir bantta silme işlemini nasıl yaptığını adım adım izah etmeye çalışalım.

Aşağıda, makinenin her adımda nasıl davranacağı bant üzerinde gösterilmiş ve altında açıklanmıştır. Sarı renge boyalı olan kutular, kafanın o anda üzerinde durduğu bant konumunu temsil etmektedir.

Netice olarak Hasan Bey'in sorusuna temel teşkil eden ve örneğin q1 üzerindeki döngülerden birisi olan b/a,R geçişi, banttan b okunduğunda banta a değerini yaz manasındadır.

SORU-7: Chomsky Hiyerarşisi (Chomsky Hierarchy) hakkında bilgi veriniz.

Bilgisayar bilimlerinin özellikle dil alanında yapılan çalışmalarında muntazam dilleri (formal languages) tasnif etmek için kullanılan bir yapıdır. Literatürde Chomsky–Schützenberger hiyerarşisi olarak da geçmektedir.

Bilindiği üzere (muntazam diller (formal langauges) veya CFG yazısından da okunabileceği üzere) muntazam dillerin dört özelliği bulunur. Bunlar özellikle içerikten bağımsız dillerin (context free languages) da temelini oluşturmuştur.

- sonlular (terminals)
- devamlılar (nonterminals)
- üretim kuralları (devamlıların değerlerini belirleyen geçiş kuralları)
- Başlangıç devamlısı

Örneğin aşağıdaki içerikten bağımsız dilbilgisini (context free grammer) ele alalım

$S \rightarrow AB$

$A \rightarrow Aa \mid \varepsilon$

$B \rightarrow b$

Yukarıdaki bu CFG tanımındaki sonlular (terminals) {a,b,ε } , devamlılar (nonterminals) { S,A,B } olarak tanımlanır. üretim kuralı olarak (production rules) S,A,B'nin açılımlarını gösteren ve → sembolü ile belirtilen satırlar bulunmaktadır. Son olarak başlangıç devamlısı (nonterminal) değeri olarak S kabul edilmiştir. Başlangıç devamlısı böyle bir kural bulunmamasına karşılık genelde S harfi (start kelimesinden gelmektedir) ile gösterilmekte ve ilk satırda bulunmaktadır.

Yukarıdaki bu CFG şayet düzenli ifadeye (regular expression) çevrilirse a*b şeklinde yazılabilir. Aslında burada anlatılan değer anb şeklinde de gösterilebilen istenildiği kadar a değeri alan (hiç almaya da bilir) ve sonra mutlaka b ile biten dildir.

Chomsky yukarıdaki şekilde tanımlanan muntazam diller (formal languages) için bir sınıflandırmaya gitmiş ve 4 seviye belirlemiştir.

- Type-0 (tip 0) Sınırlandırılmamış diller (unrestricted grammers)
- Type-1 (tip 1) İçerik duyarlı diller (context-sensitive grammers)
- Type-2 (tip 2) İçerikten bağımsız diller (context free languages)
- Type-3 (tip 3) Düzenli diller (regular grammers)

şeklinde 4 seviye isimlendirilmiştir.Bu seviyelerde iler gidildikçe (seviye arttıkça) dili bağlayan kurallarda sıkılaşmakta ve dil daha kolay işlenebilir ve daha belirgin (deterministic) bir hal almaktadır.

Tip-0 dilleri bastiçe turing makinesi (turing machine) tarafından çalıştırılabilen ve bir şekilde bitecek olan dillerdir (hesaplanabilir diller (computable languages). Örneğin özyineli sayılabilir diller (recusive enumerable languages) bu seviyeden kabul edilir.

Tip-1 dilleri için içerk duyarlı diller örnek gösterilebilir. Basitçe $\alpha A \beta \rightarrow \alpha \gamma \beta$ şeklindeki gösterime sahip bir dildir. Buradaki A devamlı (nonterminal) ve α,γ,β birer sonlu (terminal) terimdir. Bu sonlulardan α ve β boş harf (yani ε veya bazı kaynaklarda λ) olabilir ancak γ mutlak bir değere sahip olmalıdır (yani boş olamaz) Ayrıca bu tipte

$S \rightarrow \varepsilon$

şeklinde bir kurala da izin verilmektedir. Burada S devamlısının sağ tarafta olmaması gerekmektedir. Bu diller doğrusal bağlı otomatlar (linear bounded automaton) ile gösterilebilir. Doğrusal bağlı otomatlar, turing makinelerinin özel bir halidir ve Turing makinesinde bulunan bantın sabit uzunlukta olduğu (çalışmanın sabit zaman sonra sona ereceği) kabul edilir.

Tip-2 diller ise CFG ile ifade edilebilen içerikten bağımsız dillerdir (context free languages). Bu dillerin özelliği $A \rightarrow \gamma$ şeklinde gösterilmeleridir. Buradaki γ değeri sonlular (terminals)

ve devamlılar (nonterminals) olabilmektedir. Bu diller aşağı sürüklemeli otomatlar (push down automata PDA) tarafından kabul edilen dillerdir ve hemen hemen bütün programlama dillerinin temelini oluşturmaktadırlar. (programlama dillerinin neredeyse tamamı bu seviye kurallarına uymaktadırlar)

Tip-3 diller ise düzeli ifadeler (regular expressions) ile ifade edilebilen (veya üretilebilen veya parçalanabilen (parse)) dillerdir. Bu dillerin farkı

$A \rightarrow \alpha$ ve

$A \rightarrow \alpha B$

şeklindeki kurallar ile göterilebilmeleridir.

Yukarıdaki bu tanımları bir tabloda toplamak gerekirse:

Seviye	Dil Örneği	Otomat Uygulaması	Kuralları
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow \alpha$ ve $A \rightarrow \alpha B$

Yukarıdaki seviyeler bütün dilleri kapsamak için yeterli değildir ayrıca yukarıda gösterilen seviyelere giren yukarıdaki tablo dışında diller de bulunmaktadır.

SORU-8: Mana Ağları (Sematic Webs, Anlamsal Ağ) hakkında bilgi veriniz.

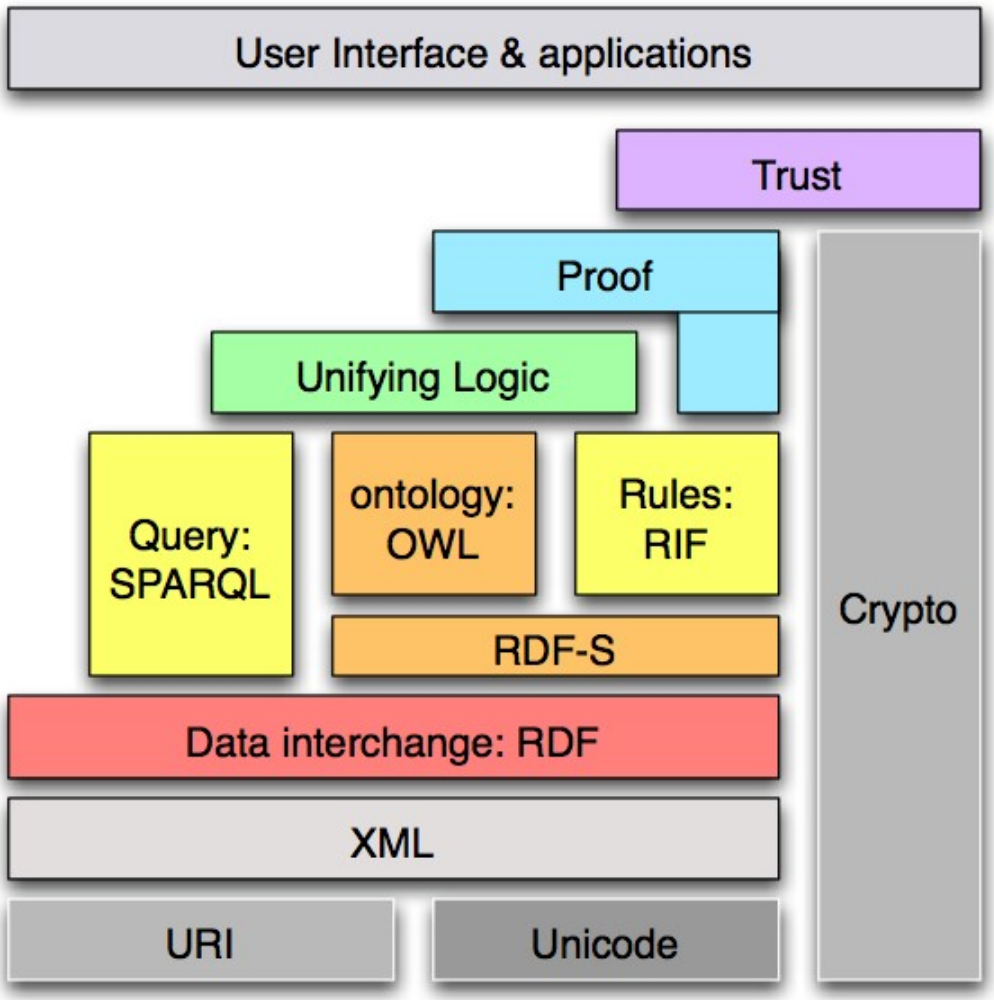
İnternetin (world wide web) bir alt uzayı olarak düşünülebilecek anlambilimsel ağlar, internet üzerinde bulunan ve doğal dilde yapılan yayınlara bir alternatiftir.

Anlambilimsel ağlar, bir bilgi kaynağının makinelere (bilgisayarlar) tarafından işlenebilecek ve bu işleme sonucunda anlamı tam olarak anlaşılabilecek ağlardır.

Mânâ ağları (anlambilimsel ağlar) üzerinde yapılan çalışmalarda henüz tam bir kesinlik yoktur. Farklı çalışma grupları, farklı standart ve alanlarda çalışmaktadırlar. Ancak Kaynak Tanım Çerçevesi (resouce description framework) ve XML kullanımları neredeyse standartlaşmıştır. Örneğin varlıkbilim (onthology) seviyesinde yapılan web ontology language (OWL) çalışması RDF kullanımınn bir örneği olarak görülebilir. Ayrıca malumat seviyesi gösterimler (knowledge representation) için çeşitli alternatifler de bulunmaktadır.

Mânâ ağlarının çıkış sebebi bilgisayarların malumat seviyesinde (knowldege) işlem yapma yetersizliğidir. Bir bilgisayarın işleyebileceği ve işlemi sonucunda çıkarabileceği anlam ile insanların anladığı anlam çok farklıdır. Örneğin "maymun" kelimesinin hangi dilden Türkçeye girdiğini araştırmak ve cevap olarak Asurcadan girdiğini bulmak insan için ulaşılabilir bir araştırma süreci iken, bilgisayarların bu konuda işlem yapmaları çelişik açıklamaları temizlemeleri ve bir malumat sahibi olmaları günümüz teknolojisinde insana göre oldukça ilkeldir. İşte bu sebeplerden oldayı İnternet üzerinde HTML ile yapılan ve

insanların okuyup anladığı sayfaların yanında bilgisayarların anlayabildiği mana ağları türemiştir.



Yukarıda W3C'nin sitesinden alınmış mana ağı yığını (semantic web stack) resmi görülmektedir. Yukarıdaki değişik katmanlarda bulunan teknolojiler w3c tarafından standartlaştırılmış teknolojilerdir.

Yukarıdaki yığına (stack) bakılınca en alt seviyede URI (Unique resource identifier) seviyesi görülmektedir. Bu seviye mana ağlarının bulunduğu servis sağlayıcılarının internet üzerindeki adresleridir. Yine alt seviyelerden Unicode ile anlatılmak istenen ise verinin işlenmesi ve iletimi sırasında kullanılacak olan kodlama standardıdır.

Örneğin XML dosyalarının saklanması ve iletimi sırasında XML dosyalarının UTF (unicode transormation table) tablolarına uygun olması istenmektedir.

Yukarıdaki şekilde dikkat edilebilecek bir nokta ise Crypto katmanının en üst seviyeye kadar çıkıyor olmasıdır. Bunun sebebi kullanılacak olan dijital imza (digital signature) ve benzeri güvenlik uygulamalarının her katmandan kontrol edilebiliyor olması ve katmanların (örneğin XML veya RDF) güvenlik parçaları barındırmamasıdır. Yani örneğin XML teknolojisinin içerisinde doğrudan bir güvenlik çözümü söz konusu değildir. Yine kullanıcı katmanı (user

interface & application) ile geri kalan mana ağı yığını (semantic web stack) arasında bulunan güven (Trust) katmanı da bu amaca hizmet etmektedir ve kullanıcının şifre katmanı (crypto) ile erişimini ayarlamaktadır.

SPARQL yine w3c tarafından standartlaştırılmış RDF sorgulama dilidir. Yani bir RDF dökümanını işleyerek sorgulamaya yarayan dillerden birsidir.

SORU-9: Meşguliyet (Utilization, Kullanım) hakkında bilgi veriniz.

Bilgisayar bilimlerinde sıra (queue) teorisinde, sıradaki bir varlığın ne kadar meşgul edildiğini ölçmeye ve bu ölçüme göre kararlar vermeye verilen isimdir.

İstatistiksel olarak ρ sembolü ile gösterilir ve şayet ρ değeri 1′den büyükse sıranın uzadığı, 1′e eşitse sıranın ne kısalıp ne de uzadığı ve şayet 1′den küçükse sıranın kısaldığı veya sırada kimse kalmadığı sonucu çıkarılır.

Burada ρ değerinin hesaplanması için aşağıdaki formül kullanılabilir:

$\rho = \lambda / \mu$

Yukarıdaki formülde λ değeri sıraya gelen varlıkların oranını, μ değeri ise sistemin bu sıradaki varlıklara hizmet verme oranını göstermektedir. Yani basitçe sıraya girenler ve sıradan çıkanların oranı olarak düşünülebilir. Daha basit bir ifadeyle talep/arz oranı olarak düşünülebilir.

Basit bir örnek üzerinden anlatmak gerekirse örneğin bir bankada bulunan banka görevlileri ortalama olarak bir müşteriye 1 dakikada hizmet veriyor olsun. Saatte 50 müşterinin geldiği bir bankada meşguliyet aşağıdaki şekilde hesaplanabilir:

Öncelikle arz ve talebi aynı birimlere getirmek gerekir. Yani bankanın arz ettiği hizmet değeri müşteri/saat şeklindeyken, bankanın talep değeri müşteri/saat cinsinden. Bu durumda bankanın arz değerini de müşteri/saat cinsine çevirelim.

Basit bir hesapla 1 dakikada bir müşteriye hizmet verilen bir bankada saatte 60 müşteriye hizmet verilir. Öyleyse bu bankanın meşguliyet oranı (utilization):

$\rho = \lambda / \mu$

$\lambda = 50$

$\mu = 60$

için $\rho = 50 / 60$

$= 0.83$

olarak bulunur. Bu durumda bankanın müşterilerine zamanında hizmet verebileceğini ve müşterilerin sıra beklemeyeceğini veya bir sıra varsa zamanla azalacağını yorumlayabiliriz.

SORU-10: EBNF (Uzatılmış BNF, Extended Backus Normal Form) hakkında bilgi veriniz.

Bilgisayar bilimlerinde dil tasarımı konusunda kullanılan backus normal şeklinin (backus normal form) özel bir halidir. Basitçe standart BNF'te yazılan kuralların birleştirilerek daha sade yazılmasını hedefler.

Bu durumu aşağıdaki örnek üzerinden görebiliriz:

Örneğin BNF olarak yazılan dilimize göre:

<EGER> ::= if(<KOSUL>) | if(<KOSUL>) else

şeklinde bir satırımız bulunsun. Bu satırın anlamı dilimizde bir EGER döz dizilimi (syntax), if komutu ve parantez içinde bir koşuldan oluşabilir veya bu if ve parantez içerisindeki koşulu bir else komutu izleyebilir.

Yukarıdaki bu BNF yazılımını EBNF olarak aşağıdaki şekilde yazabiliriz:

<EGER> ::= if(<KOSUL>) [else]

Yukarıdaki bu yeni satırda dikkat edileceği üzere köşeli parantezler arasında bir else komutu bulunmaktadır. Bunun anlamı, EGER komutu "if(KOSUL)" olarak tanımlanır ve şayet istenirse bu komuta ilave olarak else komutu eklenebilir. Yani köşeli parantez içerisindeki komut isteğe bağlıdır.

Yukarıdaki bu yeni yazılım aslında sadece gösterimde bir farklılık oluşturmaktadır. Bunun dışında, EBNF'in kullanım alanı ve işlevi BNF ile aynıdır.

EBNF'in BNF'ten farklı olarak getirdiği ifade şekilleri aşağıda listelenmiştir:

İfade		Kullanımı
Tanımlama	definition	=
Üleştirme	concatenation	,
Bitirme	termination	;
Seçim (Veya)	separation	|
Çift Tırnak	double quotation marks	" … "
Tek Tırnak	single quotation marks	' … '
İsteğe bağlı	**option**	**[...]**
Tekrarlı	**repetition**	**{ … }**
Gruplama	**grouping**	**(…)**
Yorum	**comment**	**(* … *)**
Özel dizilim	**special sequence**	**? … ?**
Hariç	**exception**	**-**

Yukarıdakitabloda ilk 6 ifade standart BNF gösteriminde de kullanılan ifadelerdir. Ancak son 6, koyu renkle yazılmış ifade EBNF için gelen yeni eklentilerdir.

Bu kullanımlardan isteğe bağlı (option) olma durumunu gördük. Şimdi diğer durumları inceleyelim:

Tekrarlı ifade ({} işaretleri arasındaki ifadeler), düzenli ifadelerde (regular expression) kullanılan * işlemine benzetilebilir. Bu işlem basitçe bir bilginin istenildiği kadar tekrar edilmesi anlamına gelir.

Örneğin programlama dillerinin çoğunda kullanılan C tipi yorum'u düşünelim (comment). Bu yorumlarda istenilen kelimeler yazılabilir. Bu durumda yorum satırının tanımı aşağıdaki şekilde olabilir

<YORUM> ::= "/*" , { <harf> } , "*/"

<harf> ::= a | b | … | z

Yukarıdaki EBNF tanımında a'dan z'ye kadar olan harfler, <harf> olarak tanımlanmış, ardından bu tanım <YORUM> içerisinde istenildiği kadar tekrarlanabilir anlamında {} işaretleri arasına yerleştirilmiştir.

EBNF'de ilave olarak dil tasarımcısının istediği yere kendi yorumlarını eklemesi de mümkündür. Buna göre tasarımcı (* *) işaretleri arasına istediği bilgiyi yazabilmektedir. Bu bilgi BNF işlemine tabi tutulmamaktadır.

EBNF'in belki de BNF'e göre en büyük eklentisi, hariç (fark) işlemidir. Yani bir bilgi grubundan başka bir bilgi grubunun çıkarılması durumudur.

Mesela bir dizgi (String) tanımı sırasında çift tırnaklar arasında herhangi bir yazı yazılabilir. Ancak bu yazının içerisinde çift tırnak bulunamaz çünkü bu durumda dizginin bittiğini belirten çift tırnak ile karışıklık oluşur. Bunu ifade için aşağıdaki EBNF gösterimini inceleyelim:

<DIZGI> ::= ' " ' , { ?bütün karakterler? – ' " ' } , ' " '

Yukarıdaki yeni kuralda, bütün karakterlerden, çift tırnak karakteri ayrı tutulmuştur. Yine yukarıdaki gösterimde ?bütün karakterler? tanımı, özel bir dizilim görüntüsüdür.

SORU-11: Eigenvalue (Özdeğer) Eigen vector (Öz yöney) Eigen Space (Öz Uzay) hakkında bilgi veriniz.

Bir yöneyin (vector) bir dönüşüme (transformation) uğramasından sonra boyutunun değişmesinden bağımsız olarak hâlâ yönü aynı kalıyorsa bu dönüşüm yöneyine (vector) öz yöney (eigen vector) ismi verilir.

Bu yön değiştirmeyen ancak uzunluk (büyüklük) değiştiren öz yöneyin yapmış olduğu değişim aslında sayısal bir uzunluk olarak hesaplanabilir (örneğin yöneyin iki misline çıkması veya yarısına inmesi gibi) işte bu hesaplanan sayısal değere (sabite, scalar) öz değer (eigen value) ismi verilir.

Yukarıdaki bu öz değerlerin (eigen value) oluşturduğu dönüşümlerin 0 olduğu matristir.

Bilgisayar bilimlerinde bir varlığın yönünü kaybetmeden değer değiştirmesi durumunda eigen öneki sıkça kullanılır. Örneğin eigenfunction (öz fonksiyon) yönü aynı olan ancak fonksiyon uygulandıktan sonra sadece miktarın değiştiği fonksiyon anlamındadır. Benzer şekilde eigenstate (özdurum), eigenmode (özhal), eigenfrequency (özfrekans, özsıklık) aynı anlamları katan ön eklerdir.

SORU-12: HTML (Hyper Text Markup Language) hakkında bilgi veriniz.

HTML, hipermetin işaretleme dili.

Kısaca İnternet sayfalarının kodlanması için kullanılan dildir. Temel olarak bütün internet tarayıcılarının desteklediği dildir. Teknoloji basitçe şu şekilde çalışır:

- HTML dilinde hazırlanmış bir dosya sunucuda bulunmaktadır
- İnternette gezen birisi bu dosyayı talep eder
- Dosya isteyen kişiye yollanır
- Dosyayı indiren kişi internet tarayıcısında bu dosyayı gösterir

Yukarıdan da anlaşılacağı üzere HTML dilini çalıştıran ve sonuçları gösteren taraf istemci (client) tarafıdır. İstemci tarafında internet tarayıcısı olarak güncel uygulamalardan herhangi birisinin (örneğin Mozilla Firefox, Internet Explorer, Opera gibi) yüklü olması yukarıdaki işlemleri yapmak için yeterlidir.

HTML dilinin standartları W3C (W 3 consortium, w3 konsorsiyumu, w3birliği) tarafından belirlenmektedir. Kısaca her web sitesinin başında olan 3 tane w yani WWW (world wide web, dünya çapında ağ) harfleri w3 olarak kısaltılmıştır. Bu birlik internet üzerinde güncel olan pekçok teknolojinin standartlarının belirlendiği ve üzerinde araştırmaların yapıldığı bir birlikteliktir ve bütün dünyadan çok sayıda şirket ve organizasyon tarafından desteklenmektedir. Birliğe www.w3c.org web adresinden erişilebilir.

Bu sitede HTML nedir sorusuna: “HTML web üzerinde yayın dilidir” şeklinde cevap verilmektedir.

Bu yazı yazılırken HTML 5.0 standartlarının yayınlandığı w3c çeşitli aralıklarla yeni eklentilerle birlikte HTML üzerinde değişiklikler yapmaktadır.

HTML dilinin gelişimine bakılacak olursa aşağıdaki adımları görebiliriz:

1991 yılında CERN’de HTML’in ilk etiketlerini (tags) içeren liste yayınlandı. 12 komuttan oluşan bu liste 1992 yılında güncellendi

1993 yılında ilk defa HTML ismi IETF (internet engineering task force) tarafından yayınlandı ve standartları belirlendi

1993 yılında yaşanan sıkıntılar sonucunda HTML+ isimli bir standart ortaya atıldıysa da 1995 yılında gelen HTML 2.0 ile bu dilde eklenti olarak gelen tablolar, metin hizalama ve figur gibi eklentiler için çözümler sunuludu

1995 yılında HTML 2.0 IETF tarafından standartlaştırıldı. Ayrıca 1997 yılına kadar, form üzerinden dosya yüklemek, tablolar, istemci tarafı resim haritaları (isamap) ve dünya çapındaki dil desteğinin arttırılması gibi eklentiler yapıldı.

1997 yılında ilk defa W3C tarafından bir standart olarak HTML 3.2 duyuruldu ve IETF , HTML ile ilgili bölümünü lağvederek bu bölümü tamamen W3C'e devretti. HTML 3.2 ile gelen yenilikler formüllerin gösterimindeki kolaylıklardı. İlk defa MathML isimli formül gösterim dili standartlaştı ayrıca yazıların yanıp sönmesi (blink) veya kayan yazı (marquee) gibi görüntüsel eklentiler yapıldı.

1997 yılının sonunda HTML 4.0 üç alternatif ile ortaya çıktı. En büyük özelliği biçim sayfalarını (style sheet) desteklemesi olan HTML 4.0 Strict ile geçmişten gelen ve Netscape ağırlıklı etiketleri tamemen yasaklıyor, Transitional ile bu etiketlere izin veriyor ve Frameset ile de sadece frame etiketlerini destekliyordu.

2000 yılında XML desteği ile birlikte XHTML dili ortaya çıktı

2008 yılının başında ise HTML 5.0 bir çalışma projesi olarak gündemde ve resmi olarak kullanılmayı bekliyor

SORU-13: Sıralama Algoritmaları (Sorting Algorithms) hakkında bilgi veriniz.

Bilgisayar bilimlerinde verilmiş olan bir grup sayının küçükten büyüğe (veya tersi) sıralanması işlemini yapan algoritmalara verilen isimdir. Örneğin aşağıdaki düzensiz sayıları ele alalım:

5 9 2 3 7 11 -4 6

Bu sayıların sıralanmış hali

-4 2 3 5 6 7 11

olacaktır. Bu sıralama işlemini yapmanın çok farklı yolları vardır ancak bilgisayar mühendisliğinin temel olarak üzerine oturduğu iki performans kriteri buradaki sonuçları değerlendirmede önemli rol oynar.

- Hafıza verimliliği (memory efficiency)
- Zaman verimliliği (Time efficiency)

Temel olarak algoritma analizindeki iki önemli kriter bunlardır. Bir algoritmanın hızlı çalışması demek daha çok hafızaya ihtiyaç duyması demektir. Tersi durumda da bir algoritmanın daha az yere ihtiyaç duyması daha yavaş çalışması demektir. Ancak bir algoritma hem zaman hem de hafıza olarak verimliyse bu durumda diğer algoritmalardan başarılı sayılabilir.

Genellikle verinin hafızada saklanması sırasında veriyi tutan bir berlirleyici özelliğinin olması istenir. Veritabanı teorisinde birincil anahtar (primary key) ismi de verilen bu özellik kullanılarak hafızada bulunan veriye erişilebilir. Bu erişme sırasında şayet berlileyici alan sıralı ise erişimin logaritmik zamanda olması mümkündür. Şayet veri sıralı değilse erişim süresi doğrusal (linear) zamanda olmaktadır.

Aşağıda bazı sıralama algoritmaları verilmiştir:

- Seçerek Sıralama (Selection Sort)
- Hızlı Sıralama Algoritması (Quick Sort Algorithm)
- Birleştirme Sıralaması (Merge Sort)
- Yığınlama Sıralaması (Heap Sort)
- Sayarak Sıralama (Counting Sort)
- Kabarcık Sıralaması (Baloncuk sıralaması, Bubble Sort)
- Taban Sıralaması (Radix Sort)
- Sokma Sıralaması (Insertion Sort)
- Sallama Sıralaması (Shaker Sort)
- Kabuk Sıralması (Shell Sort)
- Rastgele Sıralama (Bogo Sort)
- Şanslı Sıralama (Lucky Sort)
- Serseri Sıralaması (Stooge Sort)
- Şimşek Sıralaması (Flahs Sort, Bora Sıralaması)
- Tarak Sıralaması (Comb Sort)
- Gnome Sıralaması (Gnome Sort)
- Permütasyon Sıralaması (Permutation Sort)
- Strand Sort (İplik Sıralaması)

Yukarıda verilen veya herhangi başka bir sıralama algoritması genelde küçükten büyüğe doğru (ascending) sıralama yapar. Ancak bunun tam tersine çevirmek (descending) genelde algoritma için oldukça basittir. Yapılması gereken çoğu zaman sadece kontrol işleminin yönünü değiştirmektir.

Ayrıca sıralama işleminin yapılması sırasında hafızanın kullanımına göre de sıralama algoritmaları :

- Harici Sıralama (External Sort)
- Dahili Sıralama (Internal Sort)

şeklinde iki grupta incelenebilir.

Algoritmaların karşılaştırılması için aşağıdaki tablo hazırlanmıştır:

Algoritma	İngilizcesi	Algoritma Analizi			Kararlılı	Yöntem	Açıklama
		En İyi	Ortalama	En Kötü			
Seçerek Sıralama	Selection Sort	n^2	n^2	n^2	Kararsız	Seçerek	
Hızlı Sıralama	Quick Sort	n log(n)	n log(n)	n^2	Kararsız	Parçala Fethet	
Birleştirme Sıralaması	Merge Srot	n log(n)	n log(n)	n log(n)	Kararlı	Parçala Fethet	

Yığınlama Sıralaması	Heap Sort	n log(n)	n log(n)	n log(n)	Kararsız	Seçerek	
Sayarak Sıralama	Counting Sort	$n + 2^k$	$n + 2^k$	$n + 2^k$	Kararsız	Sayarak	k ikinci dizinin boyutu.
Kabarcık Sıralaması	Bubble Sort	n	n^2	n^2	Kararlı	Yer Değiştirme	
Kokteyl Sıralması	Coctail Sort	n	n^2	n^2	Kararlı	Yer Değiştirme	Çift Yönlü kabarcık sıralaması (bidirectional bubble sort) olarak da bilinir ve dizinin iki ucundan işleyen kabacrık sıralamasıdır.
Taban Sıralaması	Radix Sort	n(k/t)	n(k/t)	n(k/t)	Kararlı	Gruplama / Sayma	k, en büyük eleman değeri, t ise tabandır
Sokma Sıralaması	Insertion Sort	n	d+n	n^2	Kararlı	Sokma	d yer değiştirme sayısıdır ve n^2 cinsindendir
Sallama Sıralaması	Shaker Sort	n^2	n^2	n^2	Kararsız	Seçme	Çift yönlü seçme sıralaması (bidirectional selection sort) olarak da bilinir.
Kabuk Sıralaması	Shell Sort	$n^{3/2}$	$n^{3/2}$	$n^{3/2}$	Kararsız	Sokma	
Rastgele Sıralama	Bogo Sort	1	n n!	sonsuz	Kararsız	Rastgele	Algoritma olduğu tartışmalıdır. Knuth karıştırması (knuth shuffle) süresinde sonuca ulaşması beklenir.
Bozo Sıralaması	Bozo Sort	1	n!	sonsuz	Kararsız	Rastgele	Rastegele sıralamanın özel bir halidir.

							Rastgele olarak diziyi karıştırdıktan sonra, dizi sıralanmamışsa, yine rastgele iki sayının yeri değiştirilip denenir.
Goro Sıralaması	Goro Sort	$2^{(\log(d) / \log(2))}$	$2^{(\log(d) / \log(2))}$	$2^{(\log(d) / \log(2))}$	Kararsız	Rastgele	2011 Google kod yarışı (google code jam) sırasında ortaya çıkmıştır. Sıralanana kadar her alt küme permüte edilir. Buradaki performans değeri ispatlanmamıştır ve d dereceyi ifade eder.
Şanslı Sıralama	Lucky Sort	1	1	1	Kararsız	Rastgele	Algoritma olarak kabul edilmemelidir.
Serseri Sıralama	Stooge Sort	n^e	n^e	n^e	Kararsız	Yer değiştirme	e, doğal logairtma sayısıdır (2,71)
Şimşek Sıralaması	Partial Flash Sort	n	n + d	n + d	Kararsız	Yer Değiştirme	d, kullanılan ikinci algoritmanın performansıdır, bu algoritma bu listedekilerden herhangi birisi olabilir.
Permütasyon Sıralması	Perm Sort	n	n n!	n n!	Kararsız	Yer Değiştirme	
Bazı Yegane Sıralaması	Several Unique Sort	n	n^2	n^2	Kararsız	Yer Değiştirme	Bir bilgisayar programı tarafından bulunmuştur.
Tarak Sıralaması	Comb Sort	n log(n	n log(n)	n log(n)	Kararsız	Yer Değiştirm	Kabarcık ve hızlı sıralama

) e algoritmalarının birleşimi şeklinde düşünülebilir

Yukarıdaki yazıda geçen kararlılık kolonu ile, bir algoritmanın bitiş kontrolüne dayanmaktadır. Örneğin sıralı bir dizi verilse bile sıralama işlemi yapmaya çalışır mı?

SORU-14: TimeML hakkında bilgi veriniz.

TimeML, olaylara bağlı zaman kavramlarını ve bu zamanlar ve olaylar arasındaki ilişkileri tutmak için ağırlıklı olarak James Pustejovsky tarafından 2003 yılından beri geliştirilen XML tabanlı bir işaretleme dilidir. Dilin web üzerindeki sayfasına http://www.timeml.org adresinden erişilebilir.

Dilde temel olarak 3 seviye öğe bulunmaktadır bunlar:

1. Zaman içerikli olayların saklandığı ve ağacın yapraklarını oluşturan olaylar
2. Yaprakların üzerindeki ve zaman olayları üzerindek, zaman belirleyi değerler. Bu değerlere dilde sinyal (işaret) anlamı verilmiştir.
3. Olaylar arasındaki bağlantılar (link)

1. Zamansal Kayıtlar (yaprak seviyesi):

<EVENT>: En alt seviyedeki etiketlerdir(tag) ve olayları tutmak için kullanılır. Genel olarak her olay bir fiildir ve DTD kayıdı aşağıda verilmiştir:

```
attributes ::= eid class

eid ::= ID
{eid ::= EventID
EventID ::= e<integer>}
class ::= 'OCCURRENCE' | 'PERCEPTION'
| 'REPORTING' | 'ASPECTUAL'
| 'STATE' | 'I_STATE' | 'I_ACTION'
```

Yukarıdaki tanımdan da anlaşılacağı üzere bir olayın ayırt edici bir ID değeri bulunmalı ve bu değer bir tam sayı (integer) olmalıdır. Ayrıca bir olay yukarıda tanımlanan sınıflardan (Class) bir tanesine dahil olabilir.

<MAKEINSTANCE>: Bir olayın vücut bulduğu etikettir. Buna göre bir olay tanımı yapılmış ancak gerçekleşmemiş veya birden çok kere gerçekleşmiş olabilir. İşte her gerçekleşme durumu bir <MAKEINSTANCE> etiketidir. Bu olay nesne yönelimli programlama'daki nesne ve sınıf (object , class) ayrımı olarak düşünülebilir. Bu etiketin DTD tanımı da aşağıda verilmiştir:

```
attributes ::= eiid eventID tense aspect
[polarity] [modality] [signalID] [cardinality]
```

```
eiid ::= ID
{eiid ::= EventInstanceID
EventInstanceID ::= ei<integer>}
eventID ::= IDREF
{eventID ::= EventID}
tense ::= 'PAST' | 'PRESENT'
| 'FUTURE' | 'NONE'
aspect ::= 'PROGRESSIVE' | 'PERFECTIVE'
| 'PERFECTIVE_PROGRESSIVE' | 'NONE'
polarity ::= 'NEG' | 'POS' {default, if absent, is "POS"}
modality ::= CDATA
signalID ::= IDREF
{signalID ::= SignalID}
cardinality ::= CDATA
```

Bir <MAKEINSTANCE> etiketi basitçe bir EventID alan ve bu Event'ten bir EventInstanceID döndüren mekanizma olarak düşünülebilir. Yukarıdaki dil tanımında tense ve aspect kavramlarının karşılıkları bu dilde ne yazık ki ingilizce dilinde bulunan zamanlara göre verilmiştir. Bu durum Türkçe gibi aynı zamanları içermeyen dillerin TimeML ile modellenmesini imkansız hale getirmektedir.

Örneğin ingilizce olarak aşağıda bir zaman (tense) ve ifadeler (aspect) verilmiştir:

`tense="PAST"`

Verb group	**aspect=**
was taught	"NONE"
was being taught	"PROGRESSIVE"
had been taught	"PERFECTIVE"
had been being taught (?)	"PERFECTIVE_PROGRESSIVE"

Bu etiket üzerindeki önemli bir nokta da SignalID ve Cardinality(sayısallık) özellikleridir. Örneğin her, ençok gibi kelimeler cardinality özelliğinin değerleridir.

<TIMEX3>: Zaman ifadesdir. Setzer's (2001)'den alınan ifadeye TIMEX ve Ferro, et al. (2002)'den alınan ifadeye TIMEX2 isimleri verilmiş daha sonra yapılan güncellemeler ile bu iki ifadeden de farklı bir yapı kazandığı için TIMEX3 şeklinde isimlendirilmiştir. Kısaca bir zamanı ifade etmek için kullanılan etikettir.

```
attributes ::= tid type [functionInDocument] [beginPoint]
[endPoint] [quant] [freq] [temporalFunction]
(value | valueFromFunction) [mod] [anchorTimeID]

tid ::= ID
{tid ::= TimeID
TimeID ::= t<integer>}
type ::= 'DATE' | 'TIME' | 'DURATION' | 'SET'
beginPoint ::= IDREF
{beginPoint ::= TimeID}
endPoint ::= IDREF
```

```
{endPoint ::= TimeID}
quant ::= CDATA
freq ::= CDATA
{freq ::= duration}
functionInDocument ::= 'CREATION_TIME' | 'EXPIRATION_TIME'
| 'MODIFICATION_TIME' | 'PUBLICATION_TIME'
| 'RELEASE_TIME'| 'RECEPTION_TIME'
| 'NONE' {default, if absent, is 'NONE'}
temporalFunction ::= 'true' | 'false'
{default, if absent, is 'false'}
{temporalFunction ::= boolean}
value ::= CDATA
{value ::= duration | dateTime | time | date | gYearMonth
| gYear | gMonthDay | gDay | gMonth}
valueFromFunction ::= IDREF
{valueFromFunction ::= TemporalFunctionID
TemporalFunctionID ::= tf<integer>}
mod ::= 'BEFORE' | 'AFTER' | 'ON_OR_BEFORE' | 'ON_OR_AFTER'
|'LESS_THAN' | 'MORE_THAN'
| 'EQUAL_OR_LESS' | 'EQUAL_OR_MORE' | 'START'
| 'MID' | 'END' | 'APPROX'
anchorTimeID ::= IDREF
{anchorTimeID ::= TimeID}
```

Yukarıdaki tanımlara uyan bazı örnekler aşağıda verilmiştir:

no more than 60 days (60 günden kısa)

```
<TIMEX3 tid="t1" type="DURATION" value="P60D" mod="EQUAL_OR_LESS">
no more than 60 days
</TIMEX3>
```

the dawn of 2000 (2000'in şafağı)

```
<TIMEX3 tid="t2" type="DATE" value="2000" mod="START">
the dawn of 2000
</TIMEX3>
```

twice a month (ayda iki kere)

```
<TIMEX3 tid="t3" type="SET" value="P1M" freq="2X">
twice a month
</TIMEX3>
```

three days every month (Her ay üç gün)

```
<TIMEX3 tid="t4" type="SET" value="P1M" quant="EVERY" freq="3D">
three days every month
</TIMEX3>
```

daily (Günlük veya hergün)

```
<TIMEX3 tid="t5" type="SET" value="P1D" quant="EVERY">
daily
</TIMEX3>
```

two weeks from June 7, 2003 (7 temmuz 2003'den itibaren 2 hafta)

```
<TIMEX3 tid="t6" type="DURATION" value="P2W" beginPoint="t61"
endPoint="t62">
two weeks
</TIMEX3>
<SIGNAL sid="s1">
from
</SIGNAL>
<TIMEX3 tid="t61" type="DATE" value="2003-06-07">
```

```
June 7, 2003
</TIMEX3>
<TIMEX3 tid="t62" type="DATE" value="2003-06-21" temporalFunction="true"
anchorTimeID="t6"/>
1992 through 1995 (1992'den 1995'e kadar veya 1992 ile 1995 arası)
<TIMEX3 tid="t71" type="DATE" value="1992">
1992
</TIMEX3>
<SIGNAL sid="s1">
through
</SIGNAL>
<TIMEX3 tid="t72" type="DATE" value="1995">
1995
</TIMEX3>
<TIMEX3 tid="t7" type="DURATION" value="P4Y" beginPoint="t71"
endPoint="t72" temporalFunction="true"/>
```

2. Sinyaller (işaretler)

<SIGNAL> etiketinin amacı zamanlar üzerindeki etkilerin belirlenmesidir. Örneğin tekrarlı olaylar veya süreli olaylar gibi zaman kavramları sinyal belirterek tanımlanabilir. DTD tanımı aşağıda verilmiştir:

```
attributes ::= sid

sid ::= ID
{sid ::= SignalID
SignalID ::= s<integer>}
```

3. Bağlantılar (Links)

Birden fazla olay üzerinde bir bağlantı tanımlamaya yarar. 3 ana grupta toparlanabilirler:

- TLINK olayların bağlanmasını sağlar (örneğin “depremden sonra televizyonlar canlı yayına geçtiler”)
- SLINK ikincil veya yan cümleden gelen bağlantılardır. Örneğin “Ali eve geç geliyor, akşam yemeklerinde evde olmuyor”
- ALINK bakış ifade eden bağlantılardır. Örneğin”gemi batmaya başladı”, “kurtarma ekibi kurtulanları aramayı bıraktı”

Bu bağlantı etiketlerinin (tags) DTD tanımları aşağıda listelenmiştir:

<TLINK>

```
attributes ::= [lid] [origin] (eventInstanceID | timeID) [signalID]
(relatedToEventInstance | relatedToTime) relType

lid ::= ID
{lid ::= LinkID
LinkID ::= l<integer>}
origin ::= CDATA
eventInstanceID ::= IDREF
{eventInstanceID ::= EventInstanceID}
timeID ::= IDREF
{timeID ::= TimeID}
```

```
signalID ::= IDREF
{signalID ::= SignalID}
relatedToEventInstance ::= IDREF
{relatedToEventInstance ::= EventInstanceID}
relatedToTime ::= IDREF
{relatedToTime ::= TimeID}
relType ::= 'BEFORE' | 'AFTER' | 'INCLUDES' | 'IS_INCLUDED' | 'DURING' |
                  'SIMULTANEOUS' | 'IAFTER' | 'IBEFORE' | 'IDENTITY' |
                  'BEGINS' | 'ENDS' | 'BEGUN_BY' | 'ENDED_BY'
```

<SLINK>

```
attributes ::= [lid] [origin] [eventInstanceID] [signalID]
subordinatedEventInstance relType

lid ::= ID
{lid ::= LinkID
LinkID ::= l<integer>}
origin ::= CDATA
eventInstanceID ::= IDREF
{eventInstanceID ::= EventInstanceID}
subordinatedEventInstance ::= IDREF
{subordinatedEventInstance ::= EventInstanceID}
signalID ::= IDREF
{signalID ::= SignalID}
relType ::= 'MODAL' | 'EVIDENTIAL' | 'NEG_EVIDENTIAL'
          | 'FACTIVE' | 'COUNTER_FACTIVE'
```

<ALINK>

```
attributes ::= [lid] [origin] eventInstanceID [signalID]
relatedToEventInstance relType

lid ::= ID
{lid ::= LinkID
LinkID ::= l<integer>}
origin ::= CDATA
eventInstanceID ::= ID
{eventInstanceID ::= EventInstanceID}
signalID ::= IDREF
{signalID ::= SignalID}
relatedToEventInstance ::= IDREF
{relatedToEventInstance ::= EventInstanceID}
relType ::= 'INITIATES' | 'CULMINATES' | 'TERMINATES' | 'CONTINUES' |
'REINITIATES'
```

SORU-15: TML (Time Markup Language, Zaman İşaretleme Dili, ZİD) hakkında bilgi veriniz.

TML dili ilk defa doğal dil işleme çalışmaları sırasında bir metinde geçen olayların zaman sırasına sokulması çalışmasında doğan bir ihtiyaç üzerine XML dili üzerine inşa edilerek oluşturulmuştur. Bu dilin oluşturulmasındaki amaç göreceli olarak zamanları belirli ancak kesin olmayan olayların zamanlarının standart bir şekilde tutulmasını sağlamaktır.

Örneğin : “Ali okula geldiğinde ayşe çoktan eve gelmişti” cümlesinde “Ali’nin okula gelme” ve “Ayşenin okula gelme” olayları iki farklı olaydır. Bu cümleyi okuyan bir insan iki olayın birbirine yakın zamanlarda (tahminen saat ve dakika mertebesinde) gerçekleştiğini ancak ayşenin eve gelme olayının alinin okula gelme olayından daha önce olduğunu anlar.

TML dili tasarlanmadan önce, yukarıdaki bu yorumun standart bir dil ile gösterilmesi yapılan çalışmalarda mümkün değildir. Bu anlamda TTML, STTL, HTML+TIME ve OWL Time dilleri incelenmiş ve hiçbirisinde yukarıdaki durumun çözülemediği görülmştür.

TML dilinde olay modellemesi:

TML dilinin hedefi zamanların modellenmesidir ancak zaman modellemesine konu olan olayların modellenmesi gerekmektedir. Bu anlamda TML dili, doğal dilden cümle modeli çıkarımının (syntax analysis) basit bir örneği olarak kabul edilebilir ve bir cümlenin anlambilimsel (semantic) olarak modellenmesi olarak kabul edilebilir. Bu anlamda bir olay aşağıdaki özellikleri taşıyabilir:

Fail (Subject Phrase) : Olayı gerçekleştiren Özne.

Mefûl (Object Phrase): Olayın etkisinde olan Nesne.

Zarf (Preposition Phrase): Olayı betimleyici özellikler.

Yukarıda tanımlanan özelliklerden zarflar bilindiği üzere doğal dillerde çeşitlilik gösterebilmektedir. Örneğin yer zarfı, zaman zarfı, hal zarfı, azlık çokluk zarfı gibi zarflar bulunmaktadır. TML dilinin hedefi bütün bu zarfların modellenmesi değil ancak zaman zarflarının detaylandırılmasıdır. Dolayısıyla yukarıdaki cümle özelliklerinden zaman zarfları bu yazıda da anlatılacağı üzere detaylandırılmış diğer zarflar daha farklı çalışmalarda detaylandırılabilmek üzere bırakılmıştır.

Bir zaman zarfının özellikleri:

Bir zaman mevhumu ya kesindir yada izâfîdir (göreceli). Yani bir olayın zamanını ya saat tarih olarak tam bilmek mümkündür yada başka bir olay veya zamana göre kerteriz alarak ifade etmek mümkündür.

Örneğin günlük bir gazetede “milenyum kutlamalarında küresel barış mesajları verildi” şeklinde bir cümle geçmesi durumunda okuyucu aksi durum belirtilmediği için en yakın milenyumu ve doğal olarak 1 ocak 2000 tarihini kafasında canlandırır. Bu cümlede geçen zaman kesindir. Farklı bir örnekte “bayram dönüşü trafik yine can aldı” şeklindeki bir gazete başlığını okuyan okuyucu, doğal olarak en yakın bayram tarihini kafasında canlandırmakta ve bu tarihten daha sonraki bir tarihin izafi olarak (göreceli olarak) olayın zamanını ifade ettiğini düşünmektedir. Ayrıca farklı bir kabul ise bu tarihin günler mertebesinde olduğudur. Yani bayram dönüşü ile kastedilen zaman en fazla bayramdan 5-10 gün uzak olabilir 1-2 yıl uzak olması düşünülemez. O halde son örnekte bir olayın kesin bir zamana atıfta bulunarak zamanı verilmiştir ancak olayın kesin zamanı belirli değildir.

Kesin zamanların ifadesi:

1. Saniye (örneğin on saniye)

2. Dakika (örneğin on geçe)
3. Saat (örneğin onda)
4. Gün (örneğin onu)
5. Haftanın Günü (örneğin pazartesi)
6. Hafta (örneğin ikinci haftası)
7. Ay (örneğin mart (veya onuncu ay))
8. Yıl (örneğin 2008)

Bilgilerinden biri veya bir kaçı ile ifade edilebilir. Bu listeye ayrıca zaman dilimi eklemek mümkündür. Örneğin GMT+2 PST veya EST gibi. Ayrıca yukarıdaki listeye ifade edilmek istenen zaman detayıda eklenmelidir. Örneğin bir bayram zamanı genelde günler mertebesinde, bir ders saati genelde saatler mertebesinde veya bir öğrencinin üniversitede okuma süresi genelde yıllar mertebesindedir. Dolayısıyla ifade edilmek istenen zamanın biriminin de yukarıdaki listeye eklenmesi gerekir.
Buraya kadar olan özellikler OWL Time dilinden alınmıştır.

Süre İfadesi (Duration):

Doğal dilde kesin zamanların yanında bir olayın süresi de ifade edilebilir. Örneğin "iki saatlik toplantı" tamlamasında, toplantı süresinin başlangıcı ile bitişi arasında iki saat olduğu ifade edilmek istenmiştir. Benzer şekilde "dört yıllık fakülte" tamlamasında ise fakülte eğitiminin süresinin 4 yıl olduğu anlatılmıştır.

O halde bir olayın süresi belirtilirken aşağıdaki yapıdan faydalanılabilir.

1. Saniye (örneğin on saniyelik)
2. Dakika (örneğin on dakikalık)
3. Saat (örneğin on saatlik)
4. Gün (örneğin on günlük)
5. Hafta (örneğin iki haftalık)
6. Ay (örneğin iki aylık)
7. Yıl (örneğin dört yıllık)

Tekrarlı olaylar (Recurring events):

Doğal dilde rastlanan diğer bir zaman betimlemesi de olayın tekrarlı olması durumunda zaman bilgisinin tekrar içermesidir. Örneğin "her pazartesi" tamlamasında ifade edilen zaman pazartesidir ancak her hafta bu bilginin tekrarladığı sonucuna ulaşılır.

Tekrarlı olaylarda kesin veya muğlak bir zaman bilgisinin tekrar süresi kadar periyotta tekrarladığı anlaşılır. Bu durumda yukarıdaki bilgilere ilave olarak tekrarın periyodunu tutmak yeterlidir.

Bir tekrar periyodu aşağıdakilerden birisi veya bir kaçı ile ifade edilebilir:

1. Saniye (örneğin iki saniyede bir)
2. Dakika (örneğin onbeş dakikada bir)
3. Saat (örneğin on saatte bir)
4. Gün (örneğin beş günde bir)
5. Hafta (örneğin iki haftada bir)

6. Ay (örneğin iki ayda bir)
7. Yıl (örneğin senede bir)

Muğlak tekrarlı olaylar:

Bazı durumlarda tekrar olaylarının kesin olmaması da söz konusudur. Örneğin "Bu ilacı günde üç kere alacaksınız" veya "sabah, akşam" veya "her baharda" şeklinde ifade edilen cümlelerdeki periyotlar kesin değildir. Bu periyotların yukarıdaki kesin tekrar süreleri ile ifade edilmesi de mümkün değildir.

Muğlak zamanlar:

Bir zamanın ifadesi sırasında yukarıda belirtilen kesin yargıların kullanılması gibi kesin olmayan yargılar da bulunabilir. Örneğin "sabah toplantı var" cümlesindeki sabah kelimesinin tam olarak saat, dakika cinsinden ifadesi zordur.

Bu problemin çözümü için belirsiz kelimelerin atıfta bulundukları zaman dilimlerinin alt ve üst limitlerinin tanımlanması gerekir. Bu tanım yapılmayarak karmaşık halde bırakılması da mümkündür. Yani TML dilinin herhangi bir bilgisayar işlemesinde kullanılması sırasında insanların zaten karıştırdığı ve kesin olmayan (ambiguous) kavramların bu şekilde bırakılması veya kesinleştirilmesi kararı yazılım tasarımcısınındır. TML dilin buradaki görevi şayet kesin yargılara bağlanmak istenirse buna imkan sağlamaktır. Örneğin gece kelimesi bazı insanlar için 20.00-04.00 arası bazı insanlar için ise 24.00-08.00 arası olabilir. Bu durum zaten doğal dil kullanılırken de insanlar arasında bazı ufak karışıklıklara sebep olmaktadır.

Muğlak zamanların kesin zamanlara bağlanması için TML dilinde bir olay tanımlamak ve bu olayın süresini girmek ve tekrarını girmek yeterli olacaktır. Örneğin akşam kelimesini bir olay olarak tanımlamak ve başlangıç zamanı olarak örneğin 18.00 bitiş zamanı olarak 21.00 girmek ve tekrar süresi olarak 1 gün girmek bu kelimeyi TML üzerinde işleme imkanı sağlar.

Benzer şekilde bahar kelimesi, Mart 15 ile Haziran 15 arasıdır şeklinde tanılamak ve olayın tekrarının yıllık olduğunu ifade etmek yeterli olacaktır.

Muğlak süreler:

Doğal dillerde ifade edilen olay süreleri de muğlak olabilmektedir. Örneğin "birazdan geleceğim" veya "bir kaç saate kadar geleceğim" veya "bir iki gün içerisinde" gibi zaman zarfları kesin bir süre belirtmemektedir. Bu tip süre ifade eden zaman mevhumları için bir önceki konuda da anlatıldığı gibi istenilirse kesin zamanlara dönmek mümkündür. Sürenin başlangıcı ve bitişi için aralık vermek ve buna göre işlem yapmak TML dilinde mümkündür.

Yukarıda geçen "birazdan", "bir kaç saate kadar" veya " bir iki gün içerisinde" zarfları geleceği belirsiz zarflardır. Yani olayların başlangıcı şimdiki zamandır ve kesindir. Ancak bitişleri belirsizdir. Bu durumun tersi olarak başlangıcı belirsiz sürelerde verilebilir. Örneğin "oynamaya bir iki saat önce başladık" veya "üç dört yıldır" gibi zarflarda ise olayın başlangıç zamanını ifade eden zarfların muğlak oluşu söz konusudur.

TML dilinde muğlak süreli olayların başlangıç ve/veya bitiş zamanı için yukarıda anlatılan muğlak zaman kullanılır. Buna göre muğlak bir zaman bir aralık içerisinde kesinleştirilmiş

olur. Ancak bu durumda da dildeki karmaşa (ambiguity) kaldırılmış olur ve kelimenin farklı kişiler için ifade ettiği anlamlar kesinleştirilmiş olur.

Bir cümle yapısının özellikleri:

1. olay (eylem, fiil)
2. Özne (fail)
3. obje (meful,nesne) -> Belirtli veya Belirtisiz Nesne
4. Dolaylı Tümleç (Yüklemin yöneldiği veya bulunduğu varlık)
5. Zarf Tümleci (eylemin özelliklerini belirtir. Aşağıda alt başlıkları verilmiştir)
    1. Yer Yön (olayın geçtiği mekan)
    2. Zaman (olayın vuku bulduğu an)
    3. Ölçü (olayın miktarı)
    4. Neden (ne, ne için,ne diye)
    5. Durum (Nasıl, ne durumda, ne biçimde)
    6. Soru

Bir eylem varlığı yukarıdaki özellikleri ve alt özellikleri bünyesinde barındırabilmelidir. Bu çalışma sadece zaman mevhumuna odaklı olduğu için diğer zarf türleri sadece birer varlık olarak tanımlanacak ancak detayları bu çalışmanın dışında tutulacaktır.

Zaman Zarflarının çeşitleri. Bu çalışma sırasında görülmüştür ki zaman zarfları kesin ve bağlı (izafi) olmak üzere ikiye ayrılabilir. Bir cümlenin zamanı ifade edilirken ya kesin bir zaman ya da başka bir cümle veya olaya bağlı bir zaman bulunabilmektedir. Bu tip zamanların özellikleri yukarıda verilmiştir. Şimdi bağlı zamanların özelliklerine bakalım.

Bağlı zamanların ifadesi:

Bir bağlı zaman başka bir olaya aşağıdaki şekillerde bağlanabilir. (bu bağ şekilleri şimdiye kadar belirlenenler olup ileride belki de arttırılabilir)

İzafi zaman grubu	Açıklama	Örnek
Eş zamanlı olaylar	başlangıç ve bitiş bilinmemektedir. Olaylar aynı anda vuku bulmaktadır.	Ali oturuken aşye koşuyordu.
Sonraki olaylar	bir olay kesin olarak başka bir olayın bitmesinden sonra başlamıştır	Ali eve geldikten sonra yemeğini yedi.
Önceki olaylar	bir olay kesin olarak başka bir olayın başlamasından önce bitmiştir. Her önceki olay bir sonraki olay olarak ifade edilebilir sadece sırası farklıdır. Bu yüzden bu gruba gerek yoktur.	Ali çantasını hazırlayıp evden çıktı.
Biri önce başlamış eş zamanlı olaylar	bir olay kesin olarak başka bir olaydan önce başlamıştır ve bu iki olayın bitişi	Ali eve geldiğinde ayşe kitap okuyordu.

	bilinmemektedir	
Biri sonra başlamış eş zamanlı olaylar	Bir olay diğerine göre kesin olarak sonra başlamış ve iki olayın çakışma zamanı olan durumdur. Bu gruptaki bütün olaylar, biri önce başlamış eş zamanlı olay olarak ifade edilebilir.	Ayşe kitap okurken ali eve geldi.

Dolayısıyla olayların başlamaları, bitişleri ve sürekliliklerine göre diğer olaylara bağlanması mümkündür. O halde bir bağıl zaman ifadesinde aşağıdaki unsurlara göre bağlantı kurulabilir:

1. Başlangıç zamanı
2. Bitiş zamanı
3. Devamlılığı, Sürekliliği

TML Dilinin Söz Dizimi (Syntax):

TML Dili XML kullanılarak geliştirilmiştir aşağıda örnek bir dil yapısı bulunmaktadır:

TML Dilinde bulunan etiketler (tag):

Etiket İsmi (tag)	Üst Etiketi (parent)	Özellikleri (attributes)	Açıklama
<tml>	Null (Yok)	Null (Yok)	Dilin ana etiketidir. Bütün dil bu etiket içerisine yerleştirilir.
<event>	<tml>	Name	Bir olayı ifade etmek için kullanılır. Her olayın özellikleri bu etiket içerisinde bulunur ve isim özelliği vardır. Bu özellik içerisine eşsiz (unique) bir isim ataması yapılır.
<subject>	<event>	Name	Bir olayı gerçekleştiren faildir. Her fail olaylar arası ilişkiyi etkileyeceği için dilde eşsiz olarak isimlendirilmelidir.
<object>	<event>	Name	Bir olayın etkilediği mefuldur. Her meful olaylar arası ilişkiyi etkileyeceği için dilde eşsiz olarak isimlendirilmelidir.
<complement>	<event>	Name Type	Cümlenin tümlecidir. (dolaylı veya dolaysız ayrımı için type özelliği eklenmiştir)
<adverb>	<event>	Name Type	Her olay anlatan cümledeki her zarf için bir etiket bulunur.Eşsiz bir isme ve hangi zarf grubundan olduğunu belirten bir tipe sahiptir. Type: • Manner (Hal zarfı)

			• Location (Konum zarfı) • Frequency (Sıklık zarfı) • Time (Zaman zarfı)
<relative>	<adverb> (Type == Time)	Name Event	Sadece zaman zarflarının alt özelliği olarak kullanılırlar. Bağıl zaman belirtmeye yararlar. Altında tanımlandığı olay ile başka bir olayın göreceli zamanlarını tutacağı için ayrıca bağlanacağı bu olayın ismini alır.
<start>	<adverb> (Type == Time)	Name	Bir olayın başlangıç zamanıdır.
<end>	<adverb> (Type == Time)	Name	Bir olayın bitiş zamanıdır.
<state>	<adverb> (Type == Time)	Name	Bir olayın sürekliliğini gösterir.
<absolute>	<start> <end>	Name Obscureness	Sadece zaman zarflarınn alt özelliği olarak kullanılırlar. Kesin zaman belirtmeye yararlar. Ayrıca zamanın muğlaklığının tutulduğu bir parameter ile kesin olup olmadığı ifade edilebilir.
<duration>	<adverb> (Type == Time)	Name Obscureness	Bir sure belirtmeye yararlar. Olayın ne kadar zaman boyunca gerçekleştiğini anlatmak için kullanılırlar. Ayrıca zamanın muğlaklığının tutulduğu bir parameter ile kesin olup olmadığı ifade edilebilir.
<recurrence>	<adverb> (Type == Time)	Name Obscureness	Bir tekrar belirtmeye yararlar. Olayın ne kadar zamanla tekrar ettiğini anlatırlar. Ayrıca zamanın muğlaklığının tutulduğu bir parameter ile kesin olup olmadığı ifade edilebilir.
<second>	<absolute> <duration> <recurrence>	Value	Zaman birimleri için saniye değerini ifade eder.
<minute>	<absolute>	Value	Zaman birimleri için dakika değerini ifade eder.

	<duration> <recurrence>		
<hour>	<absolute> <duration> <recurrence>	Value	Zaman birimleri için saat değerini ifade eder.
<day>	<absolute> <duration> <recurrence>	Value	Zaman birimleri için gün değerini ifade eder.
<dayofweek>	<absolute>	Value	Zaman birimlerindne bazılarında haftanın gününü ifade eder.
<week>	<duration> <recurrence>	Value	Bazı zaman birimleri için haftalık zaman birimini ifade eder.
<month>	<absolute> <duration> <recurrence>	Value	Zaman birimleri için aylık değeri ifade eder.
<year>	<absolute> <duration> <recurrence>	Value	Zaman birimleri için yıllık değeri ifade eder.

TML Örnekleri:

Yukarıda anlatılan bilgiler ışığında ve verilen söz dizim tanımlarına göre aşağıda bir takım örnek cümleler ve bu cümlelerin TML dilindeki gösterimleri verilmiştir:

Ali okula geldi.

```
<tml>
                <event Name=olay1>
                                <subject Name=ali />
                                <adverb Name=zarf1 Type=Location>
                                 </adverb>
                </event>
</tml>
```

Ali dün okula geldi.

```
<tml>
            <event Name=olay1>
                        <subject Name=ali />
                        <adverb Name=zarf1 Type=Location>
                        </adverb>
```

```
                        <adverb Name=zarf2 Type=Time>
                                    <start Name=zaman1>
                                    </start>
                                    <end Name=zaman2>
                                    </end>
                        </adverb>
            </event>
</tml>
```

Ali hergün okula gelir.

```
<tml>
            <event Name=olay1>
                        <subject Name=ali />
                        <adverb Name=zarf1 Type=Location>
                        </adverb>
                        <adverb Name=zarf2 Type=Time>
                                    <recurrence Name = r1 Obscureness>
                                                <day value=1 />
                                    </recurrence>
                        </adverb>
            </event>
</tml>
```

SORU-16: STTL (A standard timetabling language (standart bir zaman çizelgeleme dili)) hakkında bilgi veriniz.

STTL -> A standard timetabling language (standart bir zaman çizelgeleme dili)

Bu dil ilk olarak Jeff Kingston tarafından zaman çizelgeleme problemlerine bir girdi standardı elde etmek için önerilmiştir. Dilin çıkışında kullanılan problem bir lisede haftalık ders programlarının düzgün yerleştirilmesini hedefliyordu. Nesne yönelimli (Object oriented) bir dil olarak önerilen STTL dilinde varlıklar arası miras ilişks bulunmaktadır örneğin:

```
class TEACHER inherit RESOURCE
max_class_load:INTEGER
unavailable_times:SET[TIME]
cats:SET[TEACHER_CATEGORY]
functions
defects:SEQ[DEFECT]=
(
defect_check(TeacherClash, clashes) +
defect_check(TeacherOverload, overload) +
defect_check(TeacherDayOverload, day_overload)
)
end
```

Yukarıdaki STTL alıntısında Öğretmen (TEACHER) ile kaynak (RESOURCE) arasında miras ilişkisi kurulmuştur.
Yukarıdaki tanımlamadan sonra aşağıdaki varlığın dilde tanımlanması mümkündür:

```
TEACHER("Knott", 22, {}, {English})
```

Buna göre Knott isminde bir öğretmen haftalık azami 22 saatlik ders yükü ve herhangi bir ders engeli bulunmadan English kategorisinin altında tanımlanmıştır.

STTL dilinin diğer özelliklerinden birisi de kaynak tanımlamasıdır. Örneğin dersin yapılacağı sınıfın, ders için kullanılacak malzemelerin birer kaynak olarak tanımlanması ve dolayısıyla zamanlamaya dahil edilmesi mümkündür.

```
r01= ROOM("r01", {DramaRoom})
r02 = ROOM("r02", {ScienceLab})

Yr7C = STUDENT_GROUP("Yr7C")
Yr7K = STUDENT_GROUP("Yr7K")
```

Örneğin yukarıda sınıf ve öğrenci grupları birer kaynak olarak tanımlanmıştır. Zaman modellemesi içinde alternatifler sunan STTL dilinde örneğin aşağıdaki şekilde bir zaman tanımlaması yapmak mümkündür:

```
TIME({2}, INTERVAL(1, 14), {Mon}, {INTERVAL(t(9, 30), t(12, 30))})
```

Yukarıdaki tanımlamada, 2. sömestr için 1den 14. haftalara kadar Pazartesi günleri saat 9.30 ile 12.30 arası kastedilmiştir. Dolayısıyla tekrarlı bir olayı ifade etmektedir.

Sonuç olarak zaman çizelgelemesine göre hazırlanmış olan bu dilde kesin tarihlerin belirlenmesi zaman aralıklarının ifade edilmesi mümkün olmakla beraber bağıl zaman ifadeleri ve zamanın belirsiz olması durumları düşünülmemiştir. Örneğin A ile B olayları olsun ve tek bilinenin A olayının B olayından sonra olması durumu olduğunda bunu göstermek bu dil ile mümkün değildir.

SORU-17: HTML+TIME hakkında bilgi veriniz.

HTML -> Hyper Text Markup Language (Hiper metin işaretleme dili)
TIME -> Timed Interactive Multimedia Extensions (Zaman etkileşimli çoklu ortam uzantıları)

Microsoft, Compac/DEC ve Macromedia firmaları tarafından W3C'a gönderilen bir dil önerisidir. Dil, XML üzerine kurulu SMIL (Synchronized Multimedia Integration Language , Eş güdümlü çoklu ortam uyarlama dili) üzerine kuruludur ve bu dildeki amaç ses, görüntü veya filim gibi çoklu ortam uygulamalarının web üzerinde bir standart olan HTML dilinin üzerine zamanlama ile birlikte gömülmesidir.
Buna göre örneğin Internet Gezgini 5.5 sonrasında microsoft basit bir ses kaydının sayfada istenildiği zamanda çalınmasına imkan vererebilmektedir.

Örneğin aşağıdaki kodu inceleyelim:

```
<HTML>
<HEAD>
<STYLE>
.time {behavior: url(#default#time2);}
</STYLE>
</HEAD>
<BODY>
<H1 CLASS="time" BEGIN="0″ DUR="11″ TIMEACTION="style"
STYLE="Color:Red;">timeAction</H1>
<P>Bu satır derhal görüntülenir ve altına yeni satırlar eklenir</P>
<P CLASS="time" BEGIN="2″ DUR="5″ TIMEACTION="display">2 saniye sonra
görüntülenir.</P>
```

<P CLASS=”time” BEGIN=”4″ DUR=”5″ TIMEACTION=”display”>4 saniye sonra görüntülenir.</P>
<P CLASS=”time” BEGIN=”6″ DUR=”5″ TIMEACTION=”display”>6 saniye sonra görüntülenir.</P>
<P>Son satır.</P>
</BODY>
</HTML>

Yukarıdaki kodda, bulunan 5 satırdan ilk ve son satırlar sayfanın yüklenmesinde görüntülenirken, 2. 3. ve 4. satırlar 2şer saniye ara ile görüntülenir. Bunu sağlayan P taginin içindeki sınıf stillerine (class style) ilgili zaman kodlarının yüklenmiş olmasıdır. İlginç olan diğer bir özellik ise bu yazıların sadece 5şer saniye ekranda görüntülenmeleridir.

SORU-18: OWL Time (OWL Zaman, Web Varlıkbilim Dili Zaman) hakkında bilgi veriniz.

Gelişen zamanlama ihtiyaçları ile birlikte zamanın gösterimi ve formüllenmesi de bir ihtiyaç haline gelmiştir. Örneğin yapılan her siparişte, siparişin zamanının tutulması, basir bir kiralama işleminde veya bilet satış işleminde yapılan işlemin hangi tarih ve saatler için yapıldığının tutulması artık sıradan birer gereksinim haline gelmiştir. Bu amaçla doğmuş olan OWL Time , Web Ontology Language (Baş harflerinin sırası okunmasını kolaylaştırmak için değiştirilmiş ve OWL olarak kısaltılmıştır) altında zaman göstermek amacıyla XML üzerine kurulu bir dil olarak tasarlanmıştır. Projenin daha önceki ismi DAML-Time olarak geçmekteydi.

Dilin yapısında varlıklar TemporalEntity (zaman varlığı veya fânî varlık olarak isimlendirilmektedir) ve her TemporalEntity altında Instant (kesin, ânî) veya Interval (aralık, zaman aralığı) olarak iki farklı varlık bulunur.

```
:Instant
      a       owl:Class ;
      rdfs:subClassOf :TemporalEntity .
:Interval
      a       owl:Class ;
      rdfs:subClassOf :TemporalEntity .
:TemporalEntity
      a       owl:Class ;
      rdfs:subClassOf :TemporalThing ;
      owl:equivalentClass
              [ a       owl:Class ;
                owl:unionOf (:Instant :Interval)
              ] .
```

Bu durum yukarıdaki RDF/XML gösterimi ile temsil edilmiştir. Dikkat edilirse Instant ve Interval sınıfları (Class) , TemporalEntity sınıfının (Class) birer alt sınıfıdır (subclass)

Sürelerin İfade edilmesi

OWL-Time kullanılarak sürelerin ifade edilmesi de ayrı bir problemdir. Örneğin bir eylemin süresi 1 gün ve 2 saat veya 26 saat veya 1560 dakika olabilir ve bütün bu zamanlar aynı süreye işaret etmektedir. Dolayısıyla sağlıklı bir zaman gösteriminde Yıl, Ay, Hafta, Gün, Saat, Dakika, Saniye olmak üzere 7 farklı zaman gösterimi gerekmektedir. Ayrıca bu zaman

gösterimini belirli eden ilave bir gösterici ile toplam 8lik gösterici kullanılmaktadır. Bu durum aşağıdaki sınıf yapısında gösterilmiştir:

```
:DurationDescription
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :seconds
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :minutes
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :hours
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :days
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :weeks
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :months
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :years
              ] .
```

Yukarıda gösterilen sınıf DurationDescription yani Süre Tanımlayıcı sınıfının yapısıdır. Bu sınıftan üretilen bir örnek aşağıda verilmiştir:

```
duration
      a       :DurationDescription ;
      :seconds 20 ;
      :hours 5 ;
      :days 15 .
```

Yukarıdaki örnekte 15 gün 5 saat ve 20 saniyelik bir zaman dilimi temsil edilmiştir.

Yukarıda Süre Tanılmayıcısı (DurationDescription) sınıfının tanımına dikkat edilirse her zaman birimi için ayrıca bir de maxCardinality (azami sayısallık) tanımı yapılmış ve bütün zaman birimleri için bu değer 1 atanmıştır. Bunun OWL dili açısından anlamı bu değerlerin sayısal olarak istedikleri gibi atanabilecekleridir. Ancak OWL dili bilindiği üzere varlıkbilimsel bir dildir ve her zamanın bütün bu birimleri ifade etmesi beklenemez. Örneğin üniversite eğitiminin saniyeler dakikalar cinsinden ifadesi beklenmedik bir durumdur, genelde

üniversite eğitimi 2,3,4,5,6 gibi yıllar ile ifade edilir. Veya bir yemek süresi dakika ve saatler mertebesindedir bu sürenin yıllar ile ifade edilmesi beklenmedik bir durumdur. İşte OWL-Time bu gibi durumlarda diğer zaman birimlerinin kapatılmasına imkan verir ve yapılması gereken basitçe bu zaman birimlerinin maxCardinality bilgisini 0 yapmaktır:

```
:Year
      a       owl:Class ;
      rdfs:subClassOf :DurationDescription ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality 1 ;
                owl:onProperty :years
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality 0 ;
                owl:onProperty :months
              ] ;
      ...

      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality 0 ;
                owl:onProperty :seconds
              ] .
```

Örneğin yukarıda bir yıl sınıfı tanımı yapılmıştır ve yıl dışındaki bütün zaman birimlerinin cardinality (sayısallık) bilgisi 0 yapılmıştır.

Tarih ve Saat kavramı

OWL-Time kullanılarak belirli bir tarihin veya saatin ifade edilmesi de gerekmektedir. Örneğin 10:30 bir saat ifade etmektedir buna karşılık 26 mart 2008 ise bir tarih ifade etmektedir. Dolayısıyla herhangi bir zamanı ifade etmek için öncelikle hangi birimden ifadenin yapılacağına karar verilmelidir. Yani ifade edilecek olan zaman bir dakikayı mı bir günü mü yada bir saati mi ifade etmek amacındadır belirlenmelidir. (“saat 10″ önermesi bir saat ifade ederken “26 mayıs” bir günü ifade etmektedir). Bu belirlenen birime unitType (birim tipi) denilirse geriye zaman diliminin ifade edilmesi sorunu kalmaktadır. Yani istenilen saat acaba hangi zaman dilimine göre (örneğin GMT, EST, PST) belirtilmektedir.
Son olarak tarih ve saati ifade eden zaman birimleri aşağıdaki şekilde tanımlanabilir:

```
:DateTimeDescription
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality 1 ;
                owl:onProperty :unitType
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :second
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :minute
              ] ;
```

```
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :hour
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :day
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :dayOfWeek
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :dayOfYear
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :week
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :month
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :year
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:maxCardinality 1 ;
                owl:onProperty :timeZone
              ] .

:hasDateTimeDescription
      a       owl:ObjectProperty ;
      rdfs:domain :DateTimeInterval ;
      rdfs:range :DateTimeDescription .
```

Görüldüğü üzere yukarıdaki gösterimde haftanın günleri veya yılın günü gibi kavramlar da ilave edilerek bir tarih/saat ifadesine yer verilmiştir. İlk eklenen alt sınıf (subclass) bir zaman birmini gösterirken son alt sınıf ise zaman dilimini (timeZone) göstermektedir.

Bu bilgiler paralelinde aşağıdaki toplantı varlığını inceleyelim:

```
:meetingStart
      a       :Instant ;
      :inDateTime
              :meetingStartDescription ;
      :inXSDDateTime
              2006-01-01T10:30:00-5:00 .

:meetingStartDescription
```

```
      a       :DateTimeDescription ;
      :unitType :unitMinute ;
      :minute 30 ;
      :hour 10 ;
      :day 1 ;
      :dayOfWeek :Sunday ;
      :dayOfYear 1 ;
      :week 1 ;
      :month 1 ;
      :timeZone tz-us:EST ;
      :year 2006 .
```

Yukarıda aynı toplantı varlığı için iki farklı gösterim bulunmaktadır. İlk gösterimde XSD zamanı denilen XML Scheme Definition kelimelerinin kısaltılmışı olan ve XML üzerinde tanımlama yapılan dilin biçemine uygun olarak yazılmış gösterime yer verilmiştir. Aynı tarih değeri altta bizim tanımladığımız tarih saat yapısına uygun olarak tekrar gösterilmiştri. Buna göre toplantı dakika cinsinden bir zamana işaret etmekte olup zaman birimi EST'dir ve toplantının zamanı: 1/1/2006 saat 10:30′dur.

OWL Time kullanarak zaman aralığı gösterimi
Bilindiği üzere zaman gösterimlerinde her zaman kesin ve belirli zamanları ifade etmek mümkün değildir. Örneğin bir kargo firmasının teslim süresi olarak 2-3 gün arasında demesi, teslimin en az 2 en çok 3 günde gerçekleşeceğini ifade etmektedir. Bu durum OWL Time kullanılarak aşağıdaki şekilde ifade edilebilir:

```
:DeliveryDuration
      a       owl:Class ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality 1 ;
                owl:onProperty :maxDeliveryDuration
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality 1 ;
                owl:onProperty :minDeliveryDuration
              ] .

:maxDeliveryDuration
      a       rdf:Property ;
      rdfs:domain :DeliveryDuration ;
      rdfs:range time:Interval .

:minDeliveryDuration
      a       rdf:Property ;
      rdfs:domain :DeliveryDuration ;
      rdfs:range time:Interval .
```

Yukarıdaki gösterimde bir teslim tarihi için iki alt bilgi girilmiş bunlardan birisi minDelivery (asgari teslim) diğeri ise maxDelivery (azami teslim) süresi olarak belirlenmiştir. Bu belirlemenin yanında 2 günlük ve 3 günlük sürelerde aşağıdaki şekilde tanımlanabilir:

```
:Interval2Days
      a       owl:Class ;
      rdfs:subClassOf time:Interval ;
      owl:subClassOf
              [ a       owl:Restriction ;
```

```
                owl:hasValue P2D ;
                owl:onProperty time:durationDescriptionDataType
              ] .

:Interval3Days
      a       owl:Class ;
      rdfs:subClassOf time:Interval ;
      owl:subClassOf
              [ a       owl:Restriction ;
                owl:hasValue P3D ;
                owl:onProperty time:durationDescriptionDataType
              ] .
```

Artık bu tanımlamalardan sonra kargo firmasının teslim süresi tanımlanabilir:

```
:Cargo2-3dayDuration
      a       owl:Class ;
      rdfs:subClassOf :DeliveryDuration ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom :Interval3Days ;
                owl:onProperty :maxDeliveryDuration
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:allValuesFrom :Interval2Days ;
                owl:onProperty :minDeliveryDuration
              ] .
```

yukarıda görüldüğü üzere kargo firmasının teslim süresi 2 günlük zaman diliminde asgari ve 3günlük zaman diliminde azami süreler olarak tanımlanmıştır.

OWL-Time kullanarak ne yazık ki belirsiz iki eylemin göreceli zamanlarını tanımlamak mümkün değildir. Örneğin A olayı B olayından sonra olmuştur önermesi OWL-Time kullanımıyla gösterilemez.

SORU-19: MathML (Matematiksel İşaretleme Dili, Mathematical Markup Language) hakkında bilgi veriniz.

MathML dili, genellikle internet üzerinde matematiksel formül ve sembollerin ifâde edilmesi için geliştirilmiş dildir. Dil W3C tarafından geliştirilmiş ve halen gelişmelere göre eklentiler yapılmaktadır.
Örnepin ikinci dereceden çok terimlilerin (polynom) köklerini bulmak için kullanılan delta değerini ele alalım:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Bu denklemin MathML ile gösterimi aşağıdaki şekildedir:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">

 <mi>x</mi>

 <mo>=</mo>
```

```
 <mfrac>
  <mrow>
   <mrow>
    <mo>-</mo>
    <mi>b</mi>
   </mrow>
   <mo>&PlusMinus;</mo>
   <msqrt>
    <msup>
     <mi>b</mi>
     <mn>2</mn>
    </msup>
    <mo>-</mo>
    <mrow>
     <mn>4</mn>
     <mo>&InvisibleTimes;</mo>
     <mi>a</mi>
     <mo>&InvisibleTimes;</mo>
     <mi>c</mi>
    </mrow>
   </msqrt>
  </mrow>
  <mrow>
   <mn>2</mn>
   <mo>&InvisibleTimes;</mo>
   <mi>a</mi>
  </mrow>
 </mfrac>
</math>
```

Yukarıda da görüldüğü üzere MathML dili XML'in özel bir halidir. Bu dilde bulunan her elemanın anlamı ve kullanım detayları için W3C tarafından hazırlanan siteye başvurulabilir http://www.w3.org/Math/testsuite/