4. Sözcük ve Sözdizimi Analizi

Sözdizimi analizcisi bir derleyicinin kalbidir. Çünkü diğer önemli bileşenler (anlam analizcisi ve ara kod üreticisi gibi) sözdizimi analizcisinin çıktılarına göre çalışırlar.

İlk ünitede programlama dillerinin gerçekleştirim yöntemlerinden bahsetmiştik: derleme, saf yorumlama ve hibrid gerçekleştirim. Derleme yaklaşımında, yüksek seviyeli bir dilde yazılmış programlar, derleyici adı verilen bir programla makine koduna çevrilir. Derleme yaklaşımı daha çok geniş uygulama alanlarında kullanılan programlama dillerinin gerçekleştiriminde kullanılır. Örn: C++, COBOL.

Saf yorumlama sistemlerinde kod çevrimi yapılmamaktadır; kodun orijinal hali yorumlayıcı tarafından yorumlanmaktadır. Çalışma etkinliğinin çok önemli olmadığı küçük çaplı sistemlrde tercih edilir. Örn: HTML belgelerine gömülü, script dili ile yazılmış kodlar.

Hibrid sistemler, yüksek seviyeli bir dilde yazılmış programların bir ara dile çevrilip yorumlandığı sistemlerdir. Bu sistemler script dillerinin de katkılarıyla eskiye oranla çok daha yaygın bir kullanım alanına ulaşmıştır. Derleyici sistemlerinden çok daha yavaş çalışması bu sistemlerin dezavantajıdır. Fakat son yıllarda JIT (Just in time) derleyicilerin kullanımı yaygınlaşmıştır. Örn. Java ve .NET sistemi. Bir JIT derleyicisi, bir metod ile çağrıldığında ara kodu makine koduna çevirme işlemi yapar. JIT kullanımı ile bir hibrid sistem gecikmeli derleyici sisteme dönüşmüş olur.

Tüm gerçekleştirim sistemleri hem sözcük hem de sözdizimi analizcisi kullanmaktadırlar.

Sözdizimi analizcileri (veya ayrıştırıcılar) programların sözdiziminin biçimsel tanımına dayanırlar. En yaygın olarak kullanılan sözdizimi tanımlama biçimi içerikten bağımsız dillerdir (veya BNF). BNF' in, resmi olmayan bir söz dizimi tanımlama yöntemi kullanmaya göre üç önemli avantajı vardır.
1) Hem insanlar hem de yazılımlar için BNF tanımları açık ve öz tanımlardır.
2) BNF, sözdizimi analizcisi tarafından doğrudan kullanılabilir.
3) Modüler yapıları sebebiyle BNF' I temel alan geliştirimlerin bakımı nispeten kolaydır.

Neredeyse tüm derleyiciler sözdizimi analizini iki parçaya ayırırlar: sözcük analizi ve sözdizimi analizi. Sözcük analizcisi daha küçük ölçekli dil yapıları ile ilgilenir (isimler, sayılar gibi). Sözdizimi analizcisi ise ifadeler, deyimler ve program birimleri gibi daha büyük ölçekli yapılarla ilgilenir.

Aşağıdaki üç sebepten sözcük analizi sözdizimi analizinden ayrılmaktadır:

1) Basitlik – Sözcük analizi için gereken yöntemler, sözdizimi analizi için gereken yöntemlerden daha az karmaşıktır. Dolayısıyla sözcük analizini ayrı tutarak işlemi daha basit hale getirebiliriz. Bunun yanında sözcük analizinin alt seviye detaylarını söz dizimi analizcisi hem daha küçük hem de daha az karmaşık hale getirirlir.

2) Etkinlik – Maliyetli olsa da sözcük analizcisi optimize edilebilirken, söz dizimi analizcisini optimize etmek pek anlamlı değildir.

3) Taşınabilirlik – Sözcük analizcisi girdi dosyalarını okurken buffer kullanır ve bu platform bağımlı olur. Sözdizimi analizi ise platformdan bağımsız olabilir. Bir yazılım sisteminin makine bağımlı ve makineden bağımsız parçalarını birbirinden ayrı tutmak her zaman doğru bir yaklaşımdır.

4.1. Sözcük Analizi (Lexical Analysis)

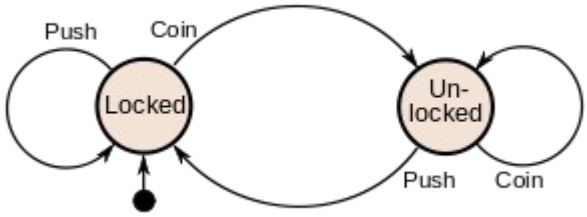
Bir sözcük analizcisi aslında bir desen eşleyicidir. Bir desen eşleyici verilen karakter desenini verilen cümle içinde bulmaya çalışır. Desen eşlemenin ilk örneklerinden biri UNIX' in ilk sürümlerinde kullanılan ed adlı bir metin editörüdür. Sonraları Perl ve JavaScript'te kullanılmıştır. Java, C++ ve C#' ın standart sınıf kütüphaneleri aracılığıyla kullanımı mümkündür.

Sözcük analizi teknik olarak sözdizimi analizinin bir parçasıdır. Bir program, derleyici için bir karakter dizisidir. Sözcük analizcisi karakterleri mantıksal gruplara ayırır ve yapılarına göre bu gruplara iç kodlar atar. Bu gruplara lexeme demiştik. Gruplara atanan iç kodlara da token diyoruz.

Sözcük analizcileri girdi olarak aldığı metni lexeme'lere ayırır ve ilgili tokenları üretir. Sözcük analizcisi yorum satırı ve boş satır gibi programın anlamını etkilemeyen kısımları atlar. Kullanıcı tanımlı isimleri sembol tablosuna yazar ve sembol tablosu derlemenin bir sonraki adımlarında kullanılır. Son olarak sözcük analizcisi token' lardaki hataları kullanıcıya bildirir.

Sözcük analizcisi geliştirmek için 3 yaklaşım vardır. Bu yaklaşımlardan biri, dilin token desenlerini tanımlayan bir durum diyagramı çizmektir. Bir durum diyagramı (durum geçiş diyagramı) yönlü bir çizgedir. Düğümler durum isimleri ile etiketlenir. Oklar, durumlar arası geçişe sebep olacak girdiler ile etiketlenir. Oklar aynı zamanda durumdan duruma geçiş yapıldığında sözcük analizcisinin yapması gereken işi de içerebilir.

Sözcük analizcilerinde kullanılan durum diyagramları sonlu otomat (finite automata) denen matematiksel makinelerin bir sınıfının gösterimidir. Sonlu otomat, düzenli diller denen dil grubunun üyelerini tanımak için tasarlanabilir. Düzenli diller, düzenli gramerlerden türetilir. Bir dile ait token'lar düzenli bir dildir, sözcük analizcisi de sonlu otomattır.



Şekil 1. Sonlu durum makinesi (Turnike örneği)

Şekil 1'de turnikeden geçişi gösteren bir sonlu durum makinesi görülmektedir. Başlangıçta kilitli olan turnike, itildiğinde durumu değişmeyecektir. Durumunu değiştirecek şey bir jeton atılmasıdır.

Sözcük analizcisi, sembol tablosunun ilk yapımından sorumludur. Sembol tablosu derleyici için isim veritabanı gibi davranır. Sembol tablosunda, kullanıcı tanımlı isimler ve özellikleri tutulmaktadır, örneğin değişken isimleri ve tipleri gibi. İsimler sembol tablosuna genellikle sözcük analizcisi tarafından yerleştirilir. İsimlere ait özellikleri sözcük analizcisini takiben çalışan derleyici bölümü yerleştirir.

4.2. Ayrıştırma Problemi (Parsing Problem)

Sözdizimi analizi olarak bildiğimiz kısım aynı zamanda ayrıştırma olarak adlandırılır. Bu bölümde ayrıştırma algoritmalarının iki ana kategorisi olan yukardan aşağı (top-down) ve aşağıdan yukarı (bottom-up) ayrıştırma anlatılacaktır.

4.2.1. Ayrıştırmaya Giriş

Programlama dilleri için ayrıştırıcılar, verilen programın ayrıştırma ağacını oluştururlar. Sözdizimi analizinin iki ayrı hedefi vardır. Birincisi, programın sözdizimsel olarak doğru olup olmadığına karar vermektir. Hata bulunduğunda ne olduğunu açıklayan bir mesaj verilmeli ve hatanın tespit edildiği yere dönerek programın geri kalanı taranmalıdır.

Sözdizimi analizinin ikinci hedefi ayrıştırma ağacı oluşturmaktır. Ağaç dil çevrimi için kullanılır. Ağacı oluşturma yönüne göre ayrıştırıcılar kategorilere ayrılır. Ayrıştırıcıların iki yaygın sınıfı kökten yapraklara (top-down) ve yapraklardan köke (bottom-up) ayrıştırma yapanlardır.

4.2.2. Yukardan Aşağı (Top-Down) Ayrıştırıcılar

Yukardan aşağı ayrıştırıcı, ağacı preorder(ön sıra) ile oluşturur. Ağaçta preorder dolaşmaya kökten başlanır. Her düğüm, dallarından önce ziyaret edilir. Düğümlerin önce sol sonra sağ dalları işlem görür. Bu en soldan türetmeye karşılık gelmektedir.

Türetme açısından bir yukardan aşağı ayrıştırıcı şöyle tanımlanabilir: En sol türetmenin herhangi bir cümlesel biçimi verildiğinde, ayrıştırıcının görevi bu en sol türetmenin bir sonraki adımındaki cümlesel biçimi bulmaktır.

xAΩ cümlesel biçimin genel tanımı olsun. Burada x, terminallerden oluşan bir metin, A bir non-terminal, Ω ise hem terminal hem de non terminal içeren karışık bir metin olsun. Burada x sadece terminallerden oluştuğundan türetmenin sonraki adımında aynı kalacaktır. A ise en soldaki non-terminal olduğu için, türetmenin bir sonraki adımında tanımlarından biri ile değiştirilecek olandır.

Örn: A->bB A->cBb A-> a tanımlarından biri seçilecek. XbBΩ, xcBbΩ, xaΩ seçeneklerinden hangisinin seçileceği bir ayrıştırma problemidir. Genellikle yukardan aşağı ayrıştırıcılar girdideki bir sonraki token'a bakarak karar verirler.

En yaygın yukardan aşağı ayrıştırma algoritmaları:

- Özyinelemeli iniş (recursive descent), BNF'e dayalı söz dizimi analizcisinin kodlanmış halidir.
- Ayrıştırma tablosu

Her iki algoritma da LL algoritmalardır ve eşit güçtedirler. İçerikten bağımsız dillerin aynı alt kümeleri üzerinde çalışırlar. LL' nin ilk L' si soldan sağa taramadan ikinci L' si ise en soldan türetmeden gelmektedir.

4.3.3. Aşağıdan Yukarıya Ayrıştırıcılar

Yapraklardan başlayıp köke doğru bir ayrıştırma ağacı oluştururlar. Bu tip ayrıştırma en sağdan türetmeye karşılık gelir. Cümlesel biçim sondan başa düğrü oluşturulur. Türetme açısından aşağıdan yukarıya ayrıştırıcılar şöyle tanımlanabilir: Verilen bir Ω cümlesel biçimine ait hangi alt metnin bir gramer kuralının sağ tarafına karşılık geldiğini bulmalıdır. Bir kural türetmenin bir önceki adımındaki cümlesel biçimi elde etmek için sol tarafına indirgenebilmesidir.

S->aAc
A->aA | b

S => aAc => aaAc => aabc

Burada sol el tarafını bulmak kolaydır ancak normalde çok zor olabilir.
Aşağıdan yukarı ayrıştırma algoritmaları LR ailesindendir. L soldan sağa taramadan, R ise en sağdan türetmeden gelmektedir.