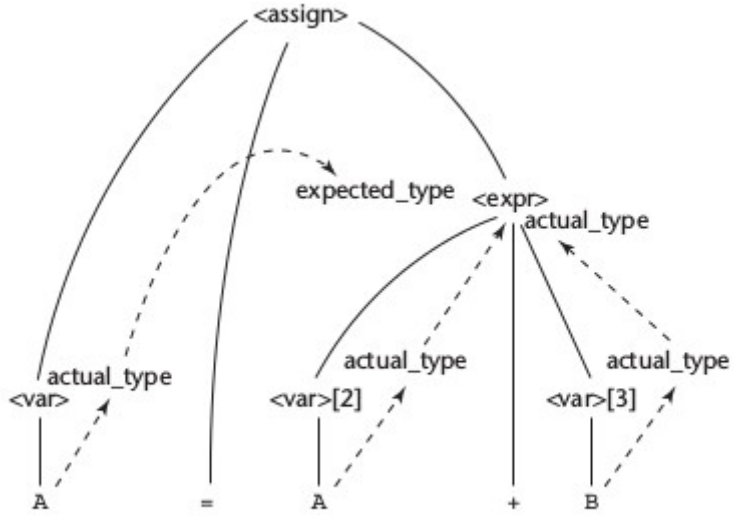
3.5. Özellik Değerlerinin Hesaplanması

Bu bölümde, ayrıştırma ağacını süslemek olarak da ifade edilen, ağacın özellik değerlerini hesaplama işlemine bakalım. Eğer bütün özellikler kalıtsal olsaydı, kökten yapraklara doğru (top-down) hesaplama yapılabilirdi. Bunun tam tersi olarak, eğer bütün özellikler sentezlenmiş olsaydı, yapraklardan köke doğru (bottom-up) hesaplama yapılabilirdi. Bizim örneğimizde hem kalıtsal hem de sentezlenmiş özellikler bulunduğu için hesaplama tek yönlü yapılmayacaktır. Aşağıda özelliklerin hangi sıra ile hesaplanacakları değerlendirilmiştir.

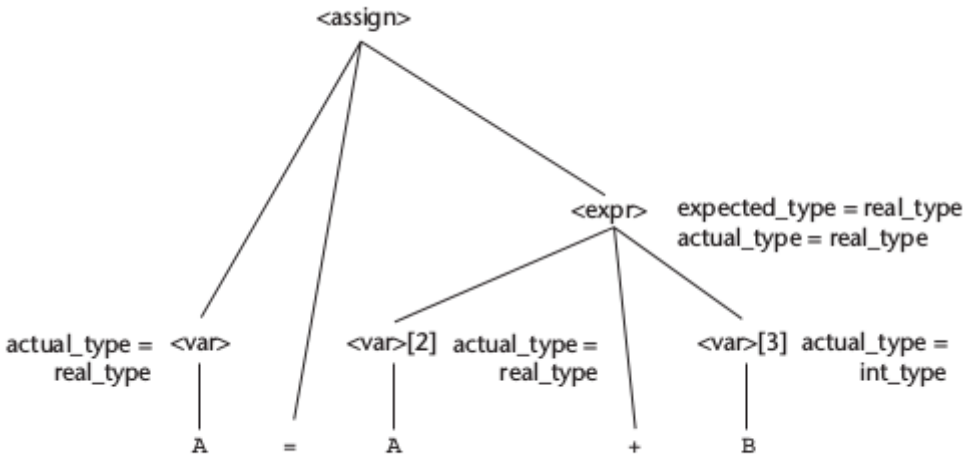
1. <var>.actual_type ← look-up(A) (Rule 4)
2. <expr>.expected_type ← <var>.actual_type (Rule 1)
3. <var>[2].actual_type ← look-up(A) (Rule 4)
   <var>[3].actual_type ← look-up(B) (Rule 4)
4. <expr>.actual_type ← either int or real (Rule 2)
5. <expr>.expected_type == <expr>.actual_type TRUE veya FALSE (Rule 2)

A=A+B ifadesi için çizmiş olduğumuz ayrıştırma ağacının özellik değerleri akışını **Şekil 1**'deki ağaçta gösterebiliriz. Ayrıştırma ağacı için düz çizgiler, özellik akışı için de kesikli çizgiler kullanılmıştır.



Şekil 1. Ağaçtaki özellik akışı

Şekil 2'deki ağaç, düğümlerdeki son özeliikleri göstermektedir. Bu ağaçlara tam özellikli (fully attributed) diyoruz. A'nın real B'nin ise int olduğu varsayılarak oluşturulmuştur.



Şekil 2. Tam özellikli ayrıştırma ağacı

Özelliklerin hangi sırada hesaplanacağı karmaşık bir problemdir ve çözümü özellik bağımlılık çizgelerinin oluşturulmasını gerektirir. Anlamsal kuralları kontrol etmek derleyicilerin ana görevidir. Tüm sözdizimi kurallarını ve anlamsal kuralları özellik gramerleri ile tanımlamak boyut ve karmaşıklık açısından zordur. Anlamsal kural ve özelliklerin sayısı arttıkça, gramerin yazılabilirliği ve okunabilirliği azalmaktadır. Dahası, geniş ayrıştırma ağaçlarında özellik hesaplamaları çok maliyetli olmaktadır.

4. Programların Anlamını Tanımlamak: Dinamik Anlamsallık

Bu bölümde, dinamik anlamsallık dediğimiz, programlama dilindeki komutların ve programların ne anlama geldiğini tanımlama konusunu göreceğiz. Sözdizimi tanımı yapmak hem doğal hem de de güçlü olan evrensel gösterim biçimi nedeniyle anlamsallığa göre daha kolay bir iştir. Dinamik anlamsallığı tanımlamak için geliştirilmiş evrensel bir gösterim biçimi veya yaklaşım bulunmamaktadır. Bu bölümde geliştirilen metotlardan birkaç tanesiniş kısaca inceleyeceğiz. Buradan sonra “dinamik anlamsallık” yerine sadece “anlamsallık” deyimini kullanacağız.

Anlamsallığı tanımlamak için bir yöntem veya gösterim biçimine olan ihtiyacın birçok farklı nedeni vardır. Yazılımcılar, programlarında kullanmadan önce, bir komutun tam olarak ne iş yaptığını bilmek zorundadır. Derleyici geliştiriciler ise doğru tasarım için dilin hangi yapılarının ne anlama geldiğini bilmelidirler. Eğer bir dilin anlamsallığı için kesin bir tanımlama bulunsaydı, o dilde yazılan programların doğruluğu, teste ihtiyaç duymadan kanıtlanmış olurdu. Sözdizimi ve anlamsallığın tam tanımlaması, bir dil için otomatik olarak derleyici oluşturacak bir araçta kullanılabilirdi.

Yazılım geliştiriciler ve derleyici tasarımcıları, programlama dilinin kullanma kılavuzundan İngilizce açıklamaları okuyarak anlamsallığa karar verirler. Bu yaklaşım pek de tatmin edici değildir çünkü bu kullanma kılavuzları genellikle tamamlanmamış olurlar ve eksik bilgi içerirler.

Programlama dillerinin anlamsallığının tanımlanma eksikliğinden dolayı, çoğu programın doğruluğu testler olmadan kanıtlanamaz. Ayrıca derleyicilerin dilin tanımlarından otomatik olarak üretilmesi mümkün değildir.

Scheme, fonksiyonel dili, resmi anlamsal tanımı olan çok az sayıdaki dilden biridir. Bu dilin kullanıldığı yaklaşıma bu bölümde değinmeyeceğiz. Bu bölüm, buyurgan (imperative) dillere uygun olan yaklaşımlara odaklanmaktadır.

4.1. İşlemsel Anlamsallık (Operational Semantics)

İşlemsel anlamsallığın altında yatan düşünce, bir program veya komutun makinede çalıştırıldığında oluşan etkileri tanımlamaktır. Makine üzerindeki etkiler makinenin durumunun (state) değişme sırası olarak izlenebilir. Burada makinenin durumu, belleğindeki değerler topluluğudur. En basit tanımıyla işlemsel anlamsallık, bir kodun derlenmiş halini bilgisayarda çalıştırmaktır.

Birçok programcı, özellikle bir dili yeni öğrenirken, bir yapının nasıl çalıştığını anlamak için küçük bir test kodu yazar. Burada programcının yaptığı, yapının anlamının ne olduğuna bakmak için işlemsel anlamsallık kullanmaktır.

Tüm dilin resmi anlamsal tanımlamasını bu yaklaşımla yapmak bir çok problemi beraberinde getirir. Makine dili ve gerçek bilgisayarlar işlemsel anlamsallık için kullanılamazlar. Daha ziyade orta-seviyeli diller ve idealize edilmiş bilgisayarlar bu işlem için özel olarak tanımlanmıştır.

4.1.1. Temel işlemler

İlk adım, ana özelliği açıklık (netlik) olan bir ara dil geliştirmektir. Bu ara dildeki her yapının açık ve belirli bir anlamı olmalıdır. Geliştirilecek dil, ara seviyede bir dildir çünkü makine dili anlaşılmak için çok düşük seviye kalmaktadır ve başka yüksek seviyeli bir dilin kullanımı da tabiki uygun olmaz.

İşlemsel anlamsallığın temel adımı yabancı bir kavram değildir. Aksine bu konu programlama kitaplarından ve bir dile ait kullanım kılavuzlarından aşina olduğumuz bir konudur. Örneğin C dilindeki for yapısı daha basit komutlar şeklinde tanımlanabilir:

C ifadesi

```
for(expr1;expr2;expr3) {
…
}
```

Anlamı

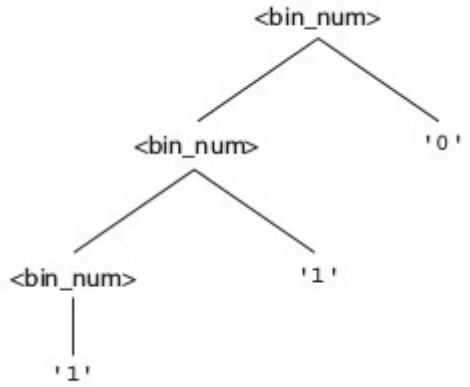
```
        expr1;
loop:   if expr2==0 goto out
        …
        expr3;
        goto loop
out:    …
```

4.2. Gösterimsel Anlamsallık (Denotational Semantics)

Bu yöntem, programların anlamını tanımlamada en yygın olarak bilinen yöntemdir. Özyinelemeli fonksiyon teorisi üzerine kurulmuştur. Bu yöntemde, dildeki herbir varlık için bir matematiksel nesne ve o varlığın örneklerini (instances) matematikselnesnenin örneklerine eşleyen bir fonksiyon tanımlanır. Yöntemin zorluğu nesneleri ve eşleme fonksiyonlarını oluşturmaktır.

Örn: İkili (binary) sayıları gösteren gramer.

<bin_num> → '0' | '1' | <bin_num> '0' | <bin_num> '1'



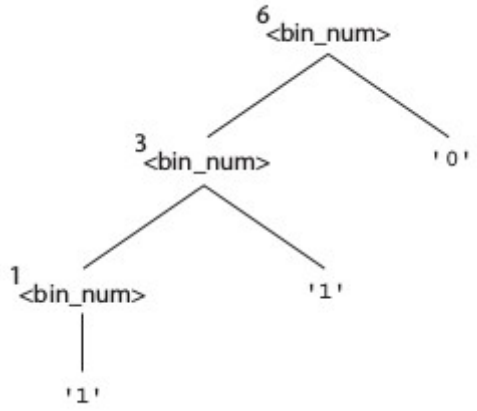
Şekil 3. 110 için ayrıştırma ağacı

Eşleme fonksiyonunun sözdizimsel domain'i tüm ikili sayıların string gösterimleri kümesidir. Anlamsal domain'i ise pozitif onlu(decimal) sayılar kümesidir (*N* ile gösterilir)

Gösterimsel anlamsallık kullanarak, ikili sayıların anlamını tanımlamak için sağ el tarafında tek terminal sembol bulunan gramer kurallarını gerçek anlamları (bu örnekte ikili sayıların onlu karşılıkları) ile birleştirmeliyiz. Örneğimizde ilk iki gramer kuralı onlu sayılarla birleştirilmelidir. Diğer iki gramer kuralında bir terminal sembolle birlikte kullanılan non-terminal görmekteyiz. Bu tip kurallara **hesaplamalı kural** diyoruz. Sağ el tarafında bir non-terminal bulunan sözdizimi kuralı, sol el tarafının anlamını hesaplayan bir fonksiyon gerektirir. Bu fonksiyon tüm sağ el tarafının anlamını ifade eder.
M_{bin} adındaki anlamsal fonksiyon sözdizimi nesneleri ile *N*'deki nesneleri birbirine eşler.

Tanım:

M_{bin} ('0') = 0
M_{bin} ('1') = 1
M_{bin} (<bin_num> '0') = 2 * M_{bin} (<bin_num>)
M_{bin} (<bin_num> '1') = 2 * M_{bin} (<bin_num>) + 1



Şekil 3. 110 için gösterimsel nesneleri ile birlikte ayrıştırma ağacı

Örn: Onlu sayılar için sözdizimsel domain 0-9 arası karakterlerdir. Anlamsal domain ise bir önceki örnekte olduğu gibi *N* kümesidir.

Onlu sayılar için sözdizimi kuralları:
<dec_num> → '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7''8'|'9'
| <dec_num> ('0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9')

Bu sözdizimi kuralları için gösterimsel eşlemeler:
M_{dec} ('0') = 0, M_{dec} ('1') = 1, M_{dec} ('2') = 2, . . ., M_{dec} ('9') = 9
M_{dec} (<dec_num> '0') = 10 * M_{dec} (<dec_num>)
M_{dec} (<dec_num> '1') = 10 * M_{dec} (<dec_num>) + 1
. . .
M_{dec} (<dec_num> '9') = 10 * M_{dec} (<dec_num>) + 9

Bir programın durumu (state)

Bir programın gösterimsel anlamsallığı ideal bir bilgisayardaki durum değişiklikleri olarak tanımlanabilir. İşlemsel anlamsallığı da aynı şekilde tanımlamıştık. Gösterimsel anlamsallık programın sadece tüm değişkenlerinin değerleri olarak tanımlanabilir. Dolayısıyla gösterimsel anlamsallık, anlamı tanımlamak için programın durumunu, işlemsel anlamsallık ise bilgisayarın durumunu kullanır. Aralarındaki temel fark, işlemsel anlamsallıkta durum değişikliklerinin bir dilde kodlanmış algoritmalarla, gösterimsel anlamsallıkta ise matematiksel fonksiyonlarla tanımlanmasıdır.

4.3. Aksiyomatik (Axiomatic) Anlamsallık

Aksiyomatik anlamsallık, anlamsallık tanımlaması için kullanılan en soyut yaklaşımdır ve matematiksel mantığa dayanmaktadır. Programın anlamını doğrudan tanımlamak yerine, programla ilgili neyin ispat edilebileceğini tanımlar. Burada, anlamsallığı tanımlamada programın doğruluğunu kanıtlama yönteminin kullanıldığını hatırlamalıyız. Aksiyomatik anlamsallıkta, makinenin veya programın durumunun modeli veya program çalıştığında oluşan durum değişikliklerinin modeli yoktur. Programın anlamı, program değişkenleri ve sabitleri arasındaki ilişkiyi temel alır. Aksiyomatik anlamsallık iki ayrı uygulamaya sahiptir: program doğrulama ve programın anlamsal tanımı.