KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BELİRSİZ LİMİT İFADELERİNİN OTOMATİK ÜRETİMİ VE ADIM ADIM ÇÖZÜMÜ

YÜKSEK LİSANS TEZİ

Bilgisayar Müh. Mehmet Cemil AYDOĞDU

HAZİRAN 2016
TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BELİRSİZ LİMİT İFADELERİNİN OTOMATİK ÜRETİMİ VE ADIM ADIM ÇÖZÜMÜ

Bilgisayar Müh. Mehmet Cemil AYDOĞDU

Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsünce
"BİLGİSAYAR YÜKSEK MÜHENDİSİ"
Unvanı Verilmesi İçin Kabul Edilen Tezdir.

Tezin Enstitüye Verildiği Tarih : 09 / 05 /2016
Tezin Savunma Tarihi : 06 / 06 /2016

Tez Danışmanı : Yrd. Doç. Dr. Hüseyin PEHLİVAN

Trabzon 2016

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Bilgisayar Mühendisliği Anabilim Dalında
Mehmet Cemil AYDOĞDU Tarafından Hazırlanan

BELİRSİZ LİMİT İFADELERİNİN OTOMATİK ÜRETİMİ VE ADIM ADIM ÇÖZÜMÜ

başlıklı bu çalışma, Enstitü Yönetim Kurulunun 10 / 05 / 2016 gün ve 1652 sayılı kararıyla oluşturulan jüri tarafından yapılan sınavda
YÜKSEK LİSANS TEZİ
olarak kabul edilmiştir.

Jüri Üyeleri

Başkan : Prof. Dr. Vasif NABİYEV .....

Üye : Yrd. Doç. Dr. Hüseyin PEHLİVAN .....

Üye : Yrd. Doç. Dr. M. M. Reza Alavi MILANI .....

Prof. Dr. Sadettin KORKMAZ
Enstitü Müdürü

ÖNSÖZ

Simgesel hesaplama, hatasız matematiksel hesaplama gerektiren mühendislik ve bilimsel alanlarda kullanılmakla birlikte çalışmamızdaki gibi bilgisayar destekli matematik öğrenimine katkıda bulunabilmektedir.

Gerek YÖK'ün Ulusal Tez Merkezi'nde yapılan araştırmalarda gerekse yapılan literatür taramalarında bu konuya yönelik çalışmaların nicelik ve nitelik olarak yetersiz kalması konunun önemini daha da artırmaktadır.

Yapmış olduğum çalışmamda JavaCC otomatik kod üretim aracı kullanılarak belirsiz limit ifadelerinin ayrıştırılması belirsiz limit ifadeleri dâhil tüm limit ifadelerinin adım adım çözümü ve belirsiz limit ifadelerinin otomatik üretimi gerçekleştirilmiştir.

Bu çalışmada danışmanlığımı üstlenen değerli hocam Yrd. Doç. Dr. Hüseyin PEHLİVAN'a ilgi, alaka ve yardımlarından dolayı teşekkürü bir borç bilirim. Ayrıca her zaman yanımda olan aileme ve dostlarıma da destekleri için teşekkür ederim.

Mehmet Cemil AYDOĞDU

Trabzon 2016

TEZ ETİK BEYANNAMESİ

Yüksek Lisans Tezi olarak sunduğum “Belirsiz Limit İfadelerinin Otomatik Üretimi ve Adım Adım Çözümü” başlıklı bu çalışmayı baştan sona kadar danışmanım Yrd. Doç. Dr. Hüseyin PEHLİVAN’ın sorumluluğunda tamamladığımı, verileri/örnekleri kendim topladığımı, deneyleri/analizleri ilgili laboratuarlarda yaptığımı/yaptırdığımı, başka kaynaklardan aldığım bilgileri metinde ve kaynakçada eksiksiz olarak gösterdiğimi, çalışma sürecinde bilimsel araştırma ve etik kurallara uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul ettiğimi beyan ederim. 06/06/2016

Mehmet Cemil AYDOĞDU

İÇİNDEKİLER

Sayfa No

ÖNSÖZ III
TEZ ETİK BEYANNAMESİ IV
İÇİNDEKİLER V
ÖZET VIII
SUMMARY IX
ŞEKİLLER DİZİNİ X
TABLOLAR DİZİNİ XI
SEMBOLLER DİZİNİ XIII
1. GENEL BİLGİLER 1
1.1. Giriş 1
1.2. Literatür Araştırması 2
1.3. Limit 7
1.3.1. Limit Tanımı 8
1.3.2. Sağ Sol Limit 8
1.3.3. Limitin Çözümü 9
1.3.3.1. Grafikten Limit Bulma 9
1.3.3.2. Yerine Koyarak Limit Bulma 9
1.3.4. Limitin Kuralları 9
1.3.5. Limit Belirsizliği 10
1.3.6. Belirsiz Limit Çözümü 11
1.3.6.1. Sadeleştirme Dönüştürme 11
1.3.6.2. L'Hopital Kuralı 12
1.3.6.2.1. Türev 12
1.3.6.2.2. L'Hopital Kuralları 14
1.4. Matematiksel İfadelerin Ayrıştırılması 15
1.4.1. Sözcüksel (Lexical) Analiz 16
1.4.2. Yazım(Sentaks) Analizi 17
1.4.2.1. Ayrıştırma Ağaçları 18
1.4.2.2. Dillerin Hiyerarşisi 18
1.4.2.3. Bağlam-bağımsız Diller 19

1.4.2.4. Backus–Naur Biçimi 19
1.4.2.5. Belirsiz ve Belirli Dil Bilgisi 20
1.4.2.6. Operatör Önceliği 21
1.4.2.7. Operatör Birleşimi 21
1.4.2.8. First ve Follow Kümeleri 21
1.4.2.9. Ayrıştırma Algoritmaları 22
1.4.2.9.1. LL Ayrıştırma 22
1.4.2.9.1.1. Soldan yineleme 23
1.4.2.9.1.2. Sol Çarpanlama 24
1.4.2.9.2. LR Ayrıştırma 24
1.4.3. Anlambilimsel (Semantik) Analiz 25
1.5. Ayrıştırıcı Üreteçleri 25
1.5.1. Yacc 26
1.5.2. JavaCC 27
1.5.2.1. EBNF 28
1.5.2.2. AST 29
1.5.2.3. Ziyaretçi Tasarım Şablonu 31
1.6. Kullanılan Diğer Teknolojiler 32
1.6.1. Maven 33
1.6.2. Junit 33
1.6.3. Jsp 33
1.6.4. Tomcat 34
1.6.5. Mathjax 34
2. YAPILAN ÇALIŞMALAR, BULGULAR VE İRDELEME 35
2.1. Giriş 35
2.2. Belirsiz Limit Sorularının Adım Adım Çözümü 35
2.2.1. Limit Grameri 37
2.2.2. JavaCC Dosyası 37
2.2.3. AST Tanımlamaları 40
2.2.4. Limit Değerlendirme 42
2.2.5. Belirsizlik Tespiti 43
2.2.6. Belirsizlik Çözümü 45
2.2.7. L'Hopital Kuralının Uygulaması 46

2.2.8. Türev Alma 46
2.2.9. Sadeleştirme 48
2.2.10. Sadeleştirme Sorunları: 53
2.2.11. Trigonometrik Fonksiyonlar 55
2.2.12. L'Hopital Kuralının Sorunları 58
2.2.13. Belirsizlik Dönüştürme 59
2.2.14. İfadeyi Yazdırma 62
2.3. Belirsiz Limit İfadesi Üretme 63
2.3.1. Rastgele İfade Üretme 63
2.3.2. Dil Bilgisi Kuralı Tabanlı İfade Üretme 64
2.3.3. AST Tabanlı İfade Üretme 65
2.3.3.1. Sıfır Değerli İfade Üretmek 68
2.3.3.2. Sonsuz Değerli İfade Üretmek 70
2.3.4. İfade Düzenleme 70
2.3.5. İfade Yazdırma 72
2.4. Sistem Testleri 72
2.5. Kullanıcı Arayüzü 73
3. SONUÇLAR VE TARTIŞMA 76
4. ÖNERİLER 79
5. KAYNAKLAR 80
ÖZGEÇMİŞ

Yüksek Lisans Tezi

ÖZET

BELİRSİZ LİMİT İFADELERİNİN OTOMATİK ÜRETİMİ VE ADIM ADIM ÇÖZÜMÜ

Mehmet Cemil AYDOĞDU

Karadeniz Teknik Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Yrd. Doç. Dr. Hüseyin PEHLİVAN
2016, 83 Sayfa

Bu çalışmada, simgesel hesaplama yöntemi kullanılarak, JavaCC kod üretim aracı yardımıyla lise ve üniversite matematik konularından limit problemlerinin çözümü sunulmaktadır. JavaCC aracı genel olarak programlama dilleri için yorumlayıcılar geliştirilirken kullanılmasına rağmen, benzer biçimde matematiksel ifadelerin değerlendirilmesinde de kullanılabilmektedir. Çalışmada öncelikle limit ifadelerinin genel dilbilgisi kuralları çıkartılır. Daha sonra JavaCC aracı ile limit ifadelerini bu dilbilgisine göre ayrıştıracak kod üretilir. Bu kod yardımıyla ayrıştırılan ifadelerden Soyut Söz Dizim Ağacı (Abstract Syntax Tree, AST) oluşturulur. Son olarak ziyaretçi tasarım şablonu kullanılarak oluşturulan sınıf ile AST yorumlanarak problemin çözümü elde edilir.

Çalışmanın ikinci bölümünde ise mevcut otomatik matematiksel ifade üretme yaklaşımlarının belirsiz limit ifadeleri üretimi için uygun olmadığı öne sürülmüş ve buna yönelik rastgele AST üretmeye dayalı bir yöntem sunulmuştur.

Çalışma sembolik hesaplama ile belirsiz limit ifadelerinin adım adım çözümü ve üretilmesi olup bu açıdan bilgisayar destekli matematik öğrenimine de katkı sağlayacağı değerlendirilmektedir.

Anahtar Kelimeler: Belirsiz Limit, Sembolik Hesaplama, Rastgele Soru Üretimi, Adım Adım Soru Çözümü, Ayrıştırıcı, AST**,** Bilgisayar Destekli Eğitim.

Master Thesis

SUMMARY

AUTOMATIC GENERATION AND STEP BY STEP SOLUTION OF INDETERMINATE LIMIT EXPRESSIONS

Mehmet Cemil AYDOĞDU

Karadeniz Technical University
The Graduate School of Natural and Applied Sciences
Computer Engineering Graduate Program
Supervisor: Assoc. Prof. Dr. Hüseyin PEHLİVAN
2016, 83 Pages

In this paper, the solution of limit problems, which is an important subject of high school and university mathematics is presented by using JavaCC code generation tool and symbolic computation methods. Although JavaCC is generally used for generating programming language interpreters, in a similar way it can also be used in the evaluation of mathematical expressions. In this work, first the general grammar rules of limit expressions is extracted. Then parser code for the limit expressions is generated with JavaCC according to the grammar rules. Using the list of the tokens into which a limit expression is parsed with this code, an Abstract Syntax Tree (AST) is constructed. Finally, the solution is obtained by interpreting the AST with a class of Visitor Design Pattern.

In the second part of the study, it has been claimed that existing approaches are not suitable for the production of indeterminate limit expression and a method based on generating random AST is presented.

Finally a system for step by step solution and generating indeterminate limit expressions is proposed therefore the study can be regarded as a promising contribution to computer assisted education.

Key Words: Indeterminate Limit, Symbolic Computation, Automatic Expression Generation, Step By Step Solution, Parser, AST, Computer Assisted Education.

ŞEKİLLER DİZİNİ

Sayfa No

Şekil 1. İfade ayrıştırma ve değerlendirme süreci 16
Şekil 2. Tarayıcı ve ayrıştırıcı Süreci 18
Şekil 3. Belirsiz dilbilgisi ile üretilen ağaçlar 20
Şekil 4. Örnek 2+3*4 ifadesi için operatör önceliği 21
Şekil 5. LL ayrıştırma 23
Şekil 6. LR ayrıştırma 25
Şekil 7. Yacc ile ayrıştırıcı üretimi 26
Şekil 8. Limit ifadeleri için çözümleyici ana bileşenleri 36
Şekil 9. Oluşturulan AST sınıf diyagramı 41
Şekil 10. Örnek limit ifadesi için oluşturulan AST 42
Şekil 11. Belirsizlik çözüm şeması 45
Şekil 12. Farklı seviyedeki benzer terimler 53
Şekil 13. Rastgele AST üretme şeması 66
Şekil 14. Sistemin giriş ekranı 73
Şekil 15. Örnek $0/0$ biçimdeki bir belirsiz limit ifadesinin çözümü 74
Şekil 16. Örnek $\infty 0$ biçimdeki bir belirsiz limit ifadesinin çözümü 75
Şekil 17. Otomatik belirsiz limit ifadesi üretimi 75

TABLOLAR DİZİNİ

Sayfa No

Tablo 1. Genel limit kuralları 10
Tablo 2. Belirsiz limit formları 10
Tablo 3. Dört işlem için türev alma kuralları 13
Tablo 4. Üstel ve logaritmik fonksiyonların türevleri 14
Tablo 5. Trigonometrik fonksiyonların türevleri 14
Tablo 6. Belirsiz form dönüşümleri 15
Tablo 7. Dört işlem için sembol listesi 16
Tablo 8. Chomsky hiyerarşisi 19
Tablo 9. Belirsiz dil bilgisi örneği 20
Tablo 10. Belirli dilbilgisi örneği 20
Tablo 11. Tarayıcı ve ayrıştırıcı üreteçleri 26
Tablo 12. JavaCC yapılandırma dosyası 28
Tablo 13. Sayı ifadesi için BNF türünden dil bilgisi 28
Tablo 14. Sayı ifadesi için EBNF dil bilgisi 29
Tablo 15. Dört işlem için örnek dil bilgisi 29
Tablo 16. Dört işlem için AST sınıfları 30
Tablo 17. Dört işlem için dil bilgisi 30
Tablo 18. Visitor arayüzü 31
Tablo 19. Visitor sınıfı 32
Tablo 20. JavaCC .jj dosyası tanımlaması 37
Tablo 21. Limit ifadeleri için JavaCC jeton tanımlaması 38
Tablo 22. Limit ifadeleri için LL(1) dil bilgisi 38
Tablo 23. Limit ifadeleri için JavaCC dil bilgisi tanımlaması 39
Tablo 24. Limit ifadeleri için oluşturulan AST sınıfları 40
Tablo 25. Limit değerlendirme sınıfı 43
Tablo 26. Belirsizlik yaratan işlemler 43
Tablo 27. Belirsizlik tiplerinin tespiti 44
Tablo 28. AST üstünde L'Hopital uygulanması 46
Tablo 29. Türev alma sınıfı 47

Tablo 30. $x2$ ve xxifadesi için türev alma işlem adımları 48
Tablo 31. Basit sadeleştirme kuralları için bazı dönüşümler 49
Tablo 32. Sadeleştirme örnekleri 50
Tablo 33. Örnek sadeleştirme adımları 50
Tablo 34. Çarpma operatörlerini bulma metodu 53
Tablo 35. FindTimes metodunun örnek kullanımı 54
Tablo 36. İki ağacın eşitliğini değerlendiren metod 54
Tablo 37. Trigonometrik fonksiyonlar için sayısal ve simgesel hesaplama sonuçları 55
Tablo 38. Trigonometik değerler tablosu 56
Tablo 39. Örnek bir $0 * \infty$ belirsiz biçimi için belirsizlik dönüştürme 60
Tablo 40. Belirsizlik dönüştürme algoritması 60
Tablo 41. Örnek belirsiz biçimleri için belirsizlik dönüştürme 61
Tablo 42. İfade yazdırma sınıfı 62
Tablo 43. Basit rastgele cebirsel ifade üretme algoritması 63
Tablo 44. Java rastgele sayı üretme 64
Tablo 45. Rastgele belirsiz limit üretme algoritması 64
Tablo 46. Dilbilgisi tabanlı ifade üretme 65
Tablo 47. Örnek bir belirsiz ifade üretme aşamaları 67
Tablo 48. Sıfır sonucu veren işlemler 68
Tablo 49. Sıfır ile işleme girince 0 sonucu veren işlemler 68
Tablo 50. X ifadesine bağlı 0 üretmek 69
Tablo 51. Sıfır sonuçlu ifade üreten metot 69
Tablo 52. Sonucu sonsuz olan işlemler 70
Tablo 53. Sonsuz sonuçlu ifade üreten metot 70
Tablo 54. İfade düzenleme 71
Tablo 55. İfade düzenleme kodu 71
Tablo 56. Geliştirilen test sınıfı 72
Tablo 57. Örnek bir basit limit problemi çözüm adımları 76
Tablo 58. Örnek bir belirsiz limit problemi için çözüm adımları 76
Tablo 59. Diğer sistemlerle karşılaştırma 78

SEMBOLLER DİZİNİ

AST	Soyut Sözdizim Ağacı(Abstract Syntax Tree)
BNF	Backus–Naur Form
EBNF	Genişletilmiş (Extented) Backus–Naur Form
JAR	Java Archive
WAR	Web Application Archive
JSP	Java Server Page
BCS	Bilgisayar Cebir Sistemi

1. GENEL BİLGİLER

1.1. Giriş

Mühendislik uygulamalarında kullanılan matematiksel işlemler insan eliyle gerçekleştirilemeyecek derecede karmaşık olabilmektedir. Bu nedenden dolayı matematiksel işlemlerin çözümünü gerçekleştirebilmek için bilgisayar ortamında bir takım yöntemler geliştirilmiştir. Bu yöntemlerden bir tanesi olan sayısal yöntemler matematiksel ifadelerin çözümünde belli bir hata payı içermektedir. Bu yüzden sonuç tam olarak hesaplanamamaktadır. Sayısal yöntemlerin bu dezavantajına karşı simgesel hesaplama yöntemleri geliştirilmiştir. Simgesel hesaplama matematiksel işlemleri bilgisayar aracılığıyla hatasız bir biçimde tam yapabilme amacıyla kullanılmaktadır. Bilgisayar teknolojileri genellikle sayısal yöntemlere yönelik geliştirildiklerinden dolayı simgesel yöntemler için özel çözümlere gereksinim duyulmaktadır [1].

Günümüzde bilgisayar teknolojileri her alanda olduğu gibi eğitim alanına da girmiştir. Bilgisayar destekli eğitim bilimsel açıdan popüler konulardan biridir. Hatasız matematik işlemleri gerektiren mühendislik ve bilimsel alanlarda kullanılan simgesel hesaplama, eğitim alanında da önemli bir yere sahiptir. Simgesel hesaplama ile bu alandaki uygulamalar için hem otomatik soru üretimi hem de problemlerin çözümüne yönelik uygulanan işlem adımlarının gösterimi yapılabilmektedir. Matematiksel problemlerin çözümünü benzer şekilde yapabilen çeşitli profesyonel yazılımlar (WolframAlpha [2] vb.) geliştirilmiştir, ancak literatürde bu yazılımların kullandığı ifade değerlendirme metodolojileri konusunda yeterli bilgi verilmemiştir.

Eğitim alanında öğrencilerin konuyu kavraması açısından problem çözüm adımlarının gösteriminin önemli olduğu konulardan biri Limit hesaplamadır. Çünkü limit problemleri lise ve üniversite matematik düzeyindeki öğrenciler için anlaşılması zor ve karmaşık işlemler gerektirmektedir [3]. Bol ve farklı örnek soru çözümleri konunun anlaşılmasının pekiştirmesi açısından önemlidir.

Fakat konu ve test kitaplarında çözümlü örnek soru sayısı sınırlıdır ve öğrenciler çözemedikleri soruları veya kendi çözüm yollarının doğruluğunu bir uzman eğitimciye sorma ihtiyacı duyarlar.

Matematiksel işlemlerin bilgisayar ortamında simgesel olarak hesaplanmasında otomatik kod üretim araçları kullanılmaktadır. Otomatik kod üretim araçları biçimsel

dillerin analiz, yorumlama veya derleme süreçlerini kolaylaştırmak amacıyla geliştirilmiştir. Bu çalışmanın ilk kısmında Java programlama dili için geliştirilen JavaCC [4] kod üretim aracı kullanılarak öğrencilerin kavramakta zorlandığı limit problemlerinin simgesel hesaplama ile L'Hopital kuralına dayalı adım adım çözümü sunulmaktadır.

Çalışmada öncelikle konu anlatım kitapları, soru bankaları ve geçmiş sınavlarda sorulmuş limit soru tipleri incelenerek ifadelerin genel dil bilgisi yapısı belirlenmiştir. Ardından JavaCC ile limit ifadelerinin ayrıştırılması, değerlendirilmesi ve belirsiz limit formlarının çözülmesi için gereken çalışmalar yapılmış ve kaynaklardan elde edilen sorularla sistem test edilmiştir.

Çalışmanın ikinci kısımında ise çok sayıda hatasız, çözümlü soru üretmenin zaman alıcı bir süreç olmasından yola çıkarak bu soruna karşılık otomatik belirsiz limit ifadeleri üreten sistem tasarlanmıştır. Soru kaynaklarından belirsiz limit problemleri incelenerek belirsizlikleri ortaya çıkaran işlemler çıkarılmıştır. Sonuç olarak rastgele ayrıştırıcı ağacı üretimine dayanan bir yöntem önerilmiştir.

1.2. Literatür Araştırması

Bilgisayarın icadından beri en önemli kullanım amacı hızlı ve hatasız hesaplama yapmak olmuştur ve bu amaca yönelik çeşitli sistemler, algoritmalar, teknik ve metotlar geliştirilmiştir. Bilgisayar matematiğin gelişmesine yardımcı olduğu gibi aynı zamanda matematik alanındaki gelişmelerden de yararlanır. Bilgisayar, matematiksel ispatlar yapmaya, yeni matematiksel metot ve yöntemler geliştirmeye yardımcı olmaktadır. Bunun için matematiksel ifadelerin bilgisayar ortamında tanımlanması, problemlerin bilgisayar ile çözülmesine imkân sağlayacak algoritmaların geliştirilmesi, daha hızlı metotların bulunması bilgisayar bilimlerinin önemli bir alanı olduğu kadar matematiği de ilgilendirir.

Bilgisayar ve hesaplama alanı yöntemsel açıdan incelendiğinde genelde simgesel hesaplama ve sayısal hesaplama gibi ayrıldığı görülmektedir. Bunların yanında cebirsel hesaplama, algoritmik hesaplama, geometrik hesaplama vb. alt tür ve çeşitlerde tanımlamalar da yapılmaktadır. Sayısal hesaplamayı bir tarafa bırakarak simgesel hesaplama ile ilgili geçmişten günümüze kadar olan gelişmelere bakıldığında çalışmalar iki grup altında toplanabilir. Birincisi problemlerin bilgisayar tarafından uygulanabilecek çözüm yöntemleri ve algoritmalarının bulunması araştırmalarıdır. Bu alanda daha genel veya daha hızlı çözüm yöntemlerinin geliştirilmesine yönelik çalışmalar yapılmıştır. Diğer gruptaki çalışmalar ise bilgisayar cebir sistemi uygulamalarının geliştirilmesidir.

Matematik problemlerinin bilgisayarla çözümü için geliştirilen birçok algoritma vardır. M.Ö. 3. yüzyılda Euklides'in bulduğu tamsayıların en büyük ortak bölenlerinin bulunması algoritması bugünkü simgesel hesaplama algoritmalarının ilk ve temel örneği sayılabilir. Bunun yanında polinom denklemlerinin köklerinin bulunması, türev, integral ve diferansiyel denklemlerin algoritmik çözüm yöntemlerinin araştırılması simgesel hesaplamanın ilgi konusu olmuştur.

1967 yılında Berlekamp tarafından sonlu cisimler üzerindeki polinomların çarpanlara ayrılması için hızlı bir algoritma geliştirilmiştir [5]. Bundan önce kullanılan Schubert ve Kronecker algoritmaları bilgisayar için bile çok yavaş olması sebebiyle Berlekamp algoritması önemli bir gelişmedir [6]. Zassenhaus, Berlekamp algoritması üzerinde yaptığı çalışmalar sonucunda 1969 yılında bu algoritma ile elde edilen çarpanlardan tamsayılar üzerindeki çarpanların elde edilebileceğini gösterdi [7]. 1975-76 yıllarında Musser [8], Wang ve Rothschild [9] benzer yöntemleri çok değişkenli ve cebirsel katsayılı polinomlar için geliştirmiştir.

Yine 1969 yılında Risch üstel, logaritmik, trigonometrik ve rasyonel fonksiyonları içeren bir temel fonksiyonlar sınıfı için belirsiz integral problemlerinin algoritmik çözümünü oluşturdu [10]. Risch algoritmasının genelleştirilmesi üzerinde çalışmalar devam etmektedir. Yeni bulunan algoritmik çözüm yöntemleri farklı matematik problemlerin bilgisayar ile çözümünü mümkün kılmıştır.

Literatürde bilgisayarda hesaplama ile ilgili geliştirilen çeşitli uygulama ve teknolojiler de yer almaktadır. Bilgisayarla ilk simgesel hesap uygulamaları 1953 yılında Nolan [11] ve Kahrimanian [12] tarafından geliştirilen türev hesaplama programları sayılabilir. Algoritmik olarak türev hesaplayan bu program Assembly dilinde geliştirilmiştir.

Bunun ardından 1950'lerin sonuna doğru geliştirilen Lisp [13] programlama dili bilgisayarda matematik uygulamaları açısından büyük imkânlar sağlamıştır. Lisp dilinin sağladığı imkânlarla kişisel bilgisayarların simgesel hesaplama ile matematik problemlerini çözebileceğine ikna olunmuş ve alan 60'lı yıllarda hızlı bir gelişme sürecine girmiştir.

1961 yılında Slagle tarafından Lisp dili ile simgesel integral hesabı yapan bir program doktora çalışması olarak geliştirildi. SAINT adı verilen bu sistem üniversite birinci sınıf seviyesindeki integral problemlerini, integrali yeni öğrenen birinin kullandığı basit yöntemlerle çözmeyi amaçlamıştır [14].

60’lı yılların sonlarına doğru ilk genel amaçlı simgesel hesaplama sistemleri ortaya çıkmaya başlamıştır. 1968 yılında Reduce [15], 1970’de Macsyma [16] ve Reduce 2 [17], 1971’de ise Scratchpad [18] gibi cebir sistemleri geliştirilmiştir. Scratchpad’in bu yıllarda ortaya çıkan diğer sistemlerden farkı bilgisayarlı cebir sistemlerinde tür tanımlamaları kullanan ilk örnek olmasıdır.

1971 yılında Paul S. Wang tek değişkenli fonksiyonların limitini hesaplayacak MACSYMA için bir sistem geliştirmiştir [19]. Bu sistem rasyonel, logaritmik, trigonometrik ifadeleri desteklemekle beraber, L’Hopital kuralı, rasyonel fonksiyonların hızlı limit hesabı metodu gibi yöntemleri kullanmaktadır.

1998 yılında Conor Ryan Grame Evrimi: Trigonometrik kimlikleri çözme adlı çalışmalarında trigonometrik ifadelerin BNF gramerlerinden faydalanarak sembolik eşliklerinin bulunmasına yönelik bir çalışma yapmıştır [20]. Özellikle trigonometrik fonksiyonların eşliklerinin simgesel hesaplama için oluşturduğu problemi çözmeyi amaçlamıştır.

2002 yılında Cristian Bauer C++ ortamında GiNac adlı sembolik hesaplama çatı projesini geliştirmiştir [21]. C++ ortamı sayesinde nesne tabanlı yaklaşımın getirdiği faydalardan yararlanmak istemiş ve çalışması kullanıcılara C++ için bir çatı projesi olarak sunulmuştur. Yazılım sembolik olarak çok değişkenli ve faktör polinom işlemleri, en büyük ortak bölen hesaplama, seri açılımları ve matrislerle hesaplama yapma özelliklerine sahiptir.

2004 yılında Jacques Carette ifade sadeleştirmeyi anlama başlıklı bir çalışma yayınlamış [22]. Çalışmada matematiksel ifadeler için sadelik ve sadeleştirme tanımının net olarak yapılması gerekliliğinden hareketle, genel ifadeler için sadeleştirme kavramının resmi tanımını vermeyi amaçlanmıştır. Bu amaçla minimum tanımlama uzunluğu teorisinin bu alana adapte edilmiş bir hali kullanılmıştır.

2004 yılında Hyungju Park sembolik hesaplama ve sinyal işleme başlıklı çalışmasında sinyal işlemedeki birçok problemin cebirsel problemlere çevrilerek cebirsel ve simgesel hesaplama metotları ile çözülebileceğini belirtmiştir [23]. Çalışma simgesel hesaplamadan farklı mühendislik problemlerinin çözümünde de yararlanılabileceğini göstermesi açısından örnek sayılabilir.

2005 yılında Michael Beeson ve Freek Wiedjk sonsuz kavramının bilgisayarlı cebir sistemlerindeki anlamı başlıklı bir çalışma yapmıştır [24]. Çalışmada BCS sistemleri için sonsuz, tanımsız ve belirsiz kavramlarının birçok sorun teşkil ettiğini ve çoğu zaman

birbirleriyle karıştırıldığını ve hesaplamalarda hatalar oluşturduğu belirtilmiştir. Bu tür kavramlara karşılık sayı kümeleri altında filtreler oluşturan bir yöntem önerilmiştir.

2007 yılında Heinz Kredel Java ortamında bilgisayar cebir sistemi kütüphanesi çalışması gerçekleştirmiştir [25]. Çalışmalarını nesne yönelimli Java Bilgisayar Cebir Sistemi olarak tanımlamışlardır. Polinom ifadeleri için geliştirilen kütüphane Java ortamının sağladığı 64-bit ve paralel çalışabilme özelliğine sahip olduğu vurgulanmıştır. Çalışma modern programlama dili olan Java'nın kullanılması açısından bir örnektir.

2007 yılında Mohammed Shatnawi matematik aramaları için ayrıştırma ağaç normalizasyonu ile eşitlik tespiti çalışması yapmıştır [26]. Web verilerinde matematiksel ifadelerin artmasından dolayı bu veri tiplerine özel arama sorgularının yapılması ihtiyacı doğmuştur. Matematiksel ifadelerde birden fazla özdeşliği olabileceği durumlar mevcuttur. Bunun için çalışmada matematiksel ifadelerin ayrıştırma ağaçları üzerinde kural tabanlı normalizasyonlar yaparak özdeş ifadeler eşleştirilmiştir. Çalışmada matematiksel ifadeler için ayrıştırıcı JavaCC ayrıştırıcı üreteci ile üretilmiştir.

2008 yılında Labachev ve Loogen fonksiyonel bir dilde BCS uygulaması çalışması yapmıştır [27]. Bu çalışmada Haskell dilinin getirdiği avantaj ve dezavantajlar açıklanmış ve bazı algoritmalar için testler yapılarak sonuçları karşılaştırılmıştır.

2010 yılında Yoshinari Miyazaki matematiksel ifadeler için bir Bilgi Erişimi aracı geliştirmiştir [28]. Bu çalışmayla e-öğrenme yoluyla mühendislik eğitimine katkı sağlamayı amaçladıkları belirtilmiştir. MathML [29] tanımlaması temel alınarak yapılan çalışmada veri tabanı olarak Mysql, geliştirme dili olarak Java ve uygulama sunucusu olarak Tomcat kullanılmıştır. Matematiksel ifadelerin düzenli ifadelerle veri tabanı üzerinde arama sorgusu yapılmıştır.

2011 yılında Cramer Marcas, Naproche [30] sistemindeki sembolik matematiğin ayrıştırılması ve gramer belirsizliklerinin giderilmesi çalışması yapmıştır [31]. Girdilerin matematikçilerin kitap ve doküman yazarken kullandıkları dile en yakın şekliyle olmasına çalışılmıştır. Bu sayede anlaşılabilirliği sağlamayı amaçlamışlardır.

Rohit Singh ve arkadaşları 2012 yılında yayınladıkları çalışmada sorgu tabanlı bir otomatik cebir problemi üretme çalışması yapmışlar [32]. Çalışmada ispat problemleri için söz dizim kurallarını kullanarak problemler için şablonlar çıkarılmış ve bu şablonlar üzerinde geliştirdikleri sorgu dili ile sorgulamalar yaparak sorular üretmektedirler.

2015 yılında Fateman Lisp dilinde algoritmik türev alma çalışması yapmıştır [33]. ADIL ismi verdikleri otomatik türev alan bir program geliştirmiştir. Çalışmada Lisp dili ile

otomatik türev alma programın geliştirilmesi ve kullanımının oldukça basit olduğu belirtilmiştir. Ayrıca diğer benzer çalışmalardan farklı olarak türev almadaki alternatif yaklaşımlar, ileri farklar ve geri farklar yöntemleri ile ilgili deneyler gerçekleştirmişlerdir.

2013 yılında Yavuz TEKBAŞ "Otomatik Kod Üretim Araçları Yardımıyla Matematiksel İfadelerin Türevlerinin Hesaplanması ve Sadeleştirilmesi" başlıklı bir çalışmayı yüksek lisans tezi olarak sunmuştur [34]. Bu çalışma Java dili ile geliştirilmiş ve ifadelerin ayrıştırılması için JavaCC ayrıştırıcı üreteci kullanılmıştır. Türev alma ayrıştırma işlemi çıktı olan nesne ağacı üzerinde türev kurallarına göre düzenleme ve dönüştürmeler yaparak sağlanır. Ayrıca türev kurallarından kaynaklanan türev işlemi sonucu oluşan gereksiz ifadelerden kurtarmak için sadeleştirme uygulanmıştır.

2015 yılında Mir Mohammad Reza Alavi Milani doktora tezinde matematiksel ifadelerin adım adım değerlendirilmesine yönelik genel bir metodoloji çalışması yapmıştır [35]. Çalışmada matematiksel ifadeler için genel bir dil bilgisi çalışması yapılmış olup, özellikle polinom problemlerinin çözümü üzerinde durulmuştur. Çalışmada da Java programlama dili kullanılmış, matematiksel ifadelerin ayrıştırılması için JavaCC ile üretilen ayrıştırıcı kullanılmıştır.

Ayrıca doktora tezinin bir bölümünde de otomatik soru üretimi üzerine çalışılmıştır. Bu bölümün son kısmında da belirsiz limit problemleri üretimi için bir algoritma sunulmuştur. Bu algoritmada P(x) ve Q(x) şeklinde iki polinom oluşturulur. Bu iki polinomum 0 değerli bir (x-v) ifadesiyle çarpılması ile 0 değerli iki polinom elde edilir. Bu polinomların bir birine bölümünden oluşan f(x) fonksiyonunun limiti x'in v'ye yakınsadığı durumda belirsiz durum oluşturur.

Burada verilen şekliyle algoritma sadece 0/0 biçimdeki belirsiz limit problemleri üretmek için tasarlanmıştır. Diğer biçimlerdeki belirsizliklerin nasıl üretileceği belirtilmemiştir. Ayrıca algoritmada pay veya paydada belirsizlik oluşturabilen ifadelerin de bulunabileceği göz önüne alınmamıştır.

Günümüzde matematiksel problemlerin çözümünü sembolik olarak hesaplayan Mathematica [36] ve [37] Maple gibi genel hesaplama yazılımları da vardır. Maple simgesel ve sayısal hesaplama, veri analizi yapabilme, görselleştirme gibi özellikleri olan modern bir yazılımdır. Matematik ve mühendislik eğitimi için lise ve üniversitelerde yaygın bir kullanımı vardır. Yine benzer şekilde Matlab [38] uygulamasının da sembolik hesaplama için bir aracı vardır.

Ayrıca Mathematica alt yapısını kullanan matematiksel problemlerin adım adım çözümünü yapabilen web tabanlı bir arama motoru olan WolframAlfa [2] bu alandaki en güncel örneklerden biridir. Problemlerin adım adım çözümünü verme, grafiksel gösterim, web tabanlı olması, otomatik matematik problemleri üretebilme gibi birçok özelliği barındırır. Literatürdeki yayınlar genelde bu yazılımların kullanımına ve çıktılarına yönelik olup, hesaplama metodolojileri konusunda bilgi içermemektedir.

Sembolik hesaplama araçlarının ortaya çıkması ile birlikte, matematikçiler, bu araçları teoremlerin ispatlarını bilgisayarlarla yapmak için kullanmaya başladılar. Sonraki dönemlerde ise bu araçlar, matematik eğitimine destek olarak liselerde ve üniversitelerde kullanılmaya başlandı. Burada verilen çalışmalar dışında sembolik ve cebirsel hesaplama ile ilgili şifreleme, robotik modelleme, bilgisayar animasyonları, sinyal/görüntü işleme gibi bilgisayar bilimleri konularında birçok çalışma yapılmıştır.

Sonuç olarak simgesel hesaplama tarihi boyunca pek çok problemin çözümü bulunmuş bazı problemlerin de algoritmik olarak çözülemeyeceği ispatlanmıştır. Gerek daha genel çözümlerin bulunması, gerekse daha hızlı yöntemlerin geliştirilmesi açısından araştırmalar sürmektedir.

Bununla birlikte simgesel hesaplamanın imkânlarından faydalanılarak özellikle eğitim, bilim ve mühendislik alanında kullanılacak bilgisayar yazılımlarının geliştirilmesi için çalışmalar devam etmektedir. Günümüzde bu alandaki çalışmalar yeni dil ve teknolojilerle geliştirilmiş, kullanıcı dostu ara yüze sahip, kullanım kolaylığı sağlayan yeni algoritmaları içeren, paralel çalışabilen, sistemler geliştirme üzerine odaklandığı görülmektedir. Ayrıca sistemler sayısal, geometrik ve simgesel hesaplama yöntemlerini birlikte kullanacak şekilde geliştirilmektedir.

1.3. Limit

Limit kavramı başta matematik olmak üzere birçok bilim alanında kullanılan önemli bir kavramdır. Matematik ve mühendislik alanlarında sıkça kullanılan türev ve integral gibi işlemlerin temeli limit teorisine dayanmaktadır. Bu yüzden limit iyi tanımlanıp anlaşılması gereken konuların başında gelmektedir.

1.3.1. Limit Tanımı

$f(x)$, x değişkeninin a'ya yakın her değer için tanımlı olduğu bir fonksiyon olsun. $f(x)$ fonksiyonunun a'ya yeterince yakın olan x değişkenlerinde aldığı değer, bir L değerine oldukça yaklaşıyorsa, L değeri için $f(x)$ fonksiyonun limitidir tanımı yapılır. Bir başka deyişle "x a'ya yaklaşırken, $f(x)$ fonksiyonu L limitine yaklaşır" denir ve aşağıdaki gibi ifade edilir [39].

$$\lim_{x \to a} f(x) = L \tag{1.1}$$

Bu tanım gayri resmidir, çünkü *oldukça yakın* ve *yeterince yakın* ifadeleri açık değildir ve anlamları duruma göre değişiklik göstermektedir [40]. Aşağıdaki örnekte bu duruma ait ifadeler yer almaktadır.

Örnek:

a) f sabit fonksiyon ve $f(x) = x$ ise, herhangi bir a değeri için limiti a olur.

$$\lim_{x \to a} f(x) = \lim_{x \to a} x = a \tag{1.2}$$

$$\lim_{x \to 2} x = 2$$

b) f sabit fonksiyon ve $f(x) = k$ ise, herhangi bir a değeri için limiti k olur.

$$\lim_{x \to a} f(x) = \lim_{x \to a} k = k \tag{1.3}$$

$$\lim_{x \to 2} 3 = 3$$

1.3.2. Sağ Sol Limit

Bir $f(x)$ fonksiyonunun bir $x = a$ durumunda bir L limiti olması için sağdan ve soldan limitlerinin L'ye eşit olması gerekir.

$$\lim_{x\to a} f(x) = L \leftrightarrow \lim_{x\to a-} f(x) = L = \lim_{x\to a+} f(x) = L \quad (1.4)$$

1.3.3. Limitin Çözümü

Limit problemlerinin çözümü fonksiyon grafiği üzerinden veya değeri yerine koyarak çözülebilir.

1.3.3.1. Grafikten Limit Bulma

Grafiği kullanılarak x değişkeni a değerine yaklaşıyorken fonksiyonun sağdan ve soldan aldığı değerlerin incelenmesiyle fonksiyonun limit değeri bulunabilmektedir. Fonksiyonun sağdan ve soldan aldığı değerlerin yani sağdan soldan limitlerinin eşitliği durumunda x değişkeni a değerine yaklaşıyorken fonksiyonun bir limit değerinin olduğu söylenebilmektedir.

1.3.3.2. Yerine Koyarak Limit Bulma

Yerine koyarak limit bulma işlemi limitin bulunamadığı durumlar dışında kullanılan bir yöntemdir. Bu yöntemde fonksiyonun limit değeri, $\lim_{x\to a} f(x)$ ifadesinde x yerine a değeri koyularak $f(a)$ değerinin hesaplanmasıyla bulunur.

Örnek:

$\lim_{x\to 2}(3x - 1) = (3 * 2) - 1 = 5$

1.3.4. Limitin Kuralları

Verilen bir ifadenin limit değerinin yukarıdaki yöntemlerle bulunması her zaman kolay olmayabilmektedir. Bu nedenden dolayı işlem kolaylığı sağlamak için genel limit kuralları adı verilen bir takım kurallar kullanılarak verilen ifadenin limitin bulunması işlemi kolaylaştırılabilmektedir.

Limit fonksiyonunun özelliklerin kaynaklanan bu kurallar Tablo 1'de verilmiştir.

Tablo 1. Genel limit kuralları

Toplama Kuralı	$\lim_{x\to a}[f(x)+g(x)] = L+M$
Fark Kuralı	$\lim_{x\to a}[f(x)-g(x)] = L-M$
Çarpım Kuralı	$\lim_{x\to a} f(x)g(x) = LM$
Sabitle Çarpım Kuralı	$\lim_{x\to a} kf(x) = kL$
Bölüm Kuralı	$\lim_{x\to a} \frac{f(x)}{g(x)} = \frac{L}{M} \quad M \neq 0$
Kuvvet Kuralı	$\lim_{x\to a}[f(x)]^{m/n} = L^{m/n}$ $m\ çift\ sayı\ ise\ L > 0$ $m < 0\ ise\ L \neq 0$

1.3.5. Limit Belirsizliği

Belirsiz limit, belirsiz form adı verilen durumların varlığında ortaya çıkan bir sonuçtur. Belirsiz formlar 1, 0 ve ∞ içeren formlardır ve bu formlardan birini içeren bir limit ifadesinin limit değerinin ne olduğu bilinemez fakat bir takım dönüşümler veya L'Hopital kuralı kullanılarak hesaplanabilir. Bilinen en yaygın belirsiz formlar ve bu formlara karşılık düşen belirsiz limit örnekleri Tablo 2'deki gibidir.

Tablo 2. Belirsiz limit formları

Belirsizlik Türü	Örnek
$\left[\frac{0}{0}\right]$	$\lim_{x\to 0} \frac{\sin x}{x}$
$\left[\frac{\infty}{\infty}\right]$	$\lim_{x\to 0} \frac{\ln(1/x^2)}{\cot(x^2)}$
$[0.\infty]$	$\lim_{x\to 0+} x \ln\frac{1}{x}$
$[\infty-\infty]$	$\lim_{x\to(\pi/2)-} \left(\tan x - \frac{1}{\pi-2x}\right)$
$[0^0]$	$\lim_{x\to 0+} x^x$
$[\infty^0]$	$\lim_{x\to(\pi/2)-} (\tan x)^{\cos x}$
$[1^\infty]$	$\lim_{x\to\infty} \left(1+\frac{x}{x}\right)^x$

1.3.6. Belirsiz Limit Çözümü

Belirsiz limit problemleri limit fonksiyonu üzerinde belirsizliği giderecek sadeleştirme ve dönüşümler yaparak veya L'Hopital teoremi uygulayarak çözülür.

1.3.6.1. Sadeleştirme Dönüştürme

Sadeleştirme-Dönüştürme yöntemi belirsizlik durumuna yol açan terimlerin yok edilmesi işlemidir. Genelde en çok karşılaşılan $[0/0]$ belirsizlik formuna sahip limit ifadelerinde kullanılmaktadır. Limiti belirsizlikten kurtarmak için sıfır olan bölenler cebirsel olarak yok edilerek limit değeri elde edilmektedir. Bu işlem iki şekilde gerçekleşebilir:

1) Ortak böleni sadeleştirmek:

Pay ve paydada ortak bölen içeren limit ifadelerinde ortak bölenin ortadan kalkması ile belirsizlik durumu ortadan kaldırılıp, ifadenin limit değeri elde edilebilmektedir.

Örnek:

$$\lim_{x\to 1}\frac{x^2-1}{x^2-x}=\frac{0}{0}$$

Çözüm:

$$\lim_{x\to 1}\frac{x^2-1}{x^2-x}=\lim_{x\to 1}\frac{(x-1)(x+1)}{x(x-1)}=\lim_{x\to 1}\frac{x+1}{x}=\frac{2}{1}=2$$

2) Ortak çarpan yaratmak ve sadeleştirmek:

Limit ifadelerinde pay ve paydaya ortak bir çarpan eklenip belirsizlik durumu ortadan kaldırılabilmektedir.

Örnek:

$$\lim_{x\to 0}\frac{\sqrt{x+1}-\sqrt{1}}{x}=\frac{0}{0}$$

Çözüm:

$$\lim_{x\to 0}\frac{\sqrt{x+1}-\sqrt{1}}{x}=\lim_{x\to 0}\frac{\left(\sqrt{x+1}-\sqrt{1}\right)\left(\sqrt{x+1}+\sqrt{1}\right)}{x\left(\sqrt{x+1}+\sqrt{1}\right)}=\lim_{x\to 0}\frac{x+1-1}{x\left(\sqrt{x+1}+\sqrt{1}\right)}$$

$$\lim_{x\to 0}\frac{x}{x\left(\sqrt{x+1}+\sqrt{1}\right)}=\lim_{x\to 0}\frac{1}{\sqrt{x+1}+\sqrt{1}}=\frac{1}{2}$$

1.3.6.2. L'Hopital Kuralı

L'Hopital, İsviçreli matematikçi Johann Bernoulli'nin ortaya attığı, adını Fransız matematikçi Guillaume de l'Hôpital'den alan ve belirsiz forma sahip fonksiyonların limiti türev kullanarak hesaplamaya yarayan bir yöntemdir. Çoğu $[0/0]$ tipindeki belirsiz formlar basit cebir işlemleri ile (sadeleştirme, dönüştürme) çözülebilmekteyken, L'Hopital Kuralı ile $[0/0]$ belirsiz formunun dışında $[\infty/\infty]$ tipindeki limit belirsizlikleri de çözülebilmektedir. Bunlar dışındaki belirsizlik tipleri ise cebirsel dönüşüm veya logaritma alma işlemi ile bu ikisinden birine dönüştürülebilir.

L'Hopital Kuralı limit değerini hesaplarken türev işleminden yararlanır. Bunun için L'Hopital Kuralı'ndan önce türevi ele almak gerekir.

1.3.6.2.1. Türev

Türev bir fonksiyonun birim zamanda değişimini ifade etmektedir. Matematiksel anlamda türev tanımı şu şekildedir:

$A \in R$

$F = \{(x,y)|y = f(x), x \in A\}$

$$a \in A \; için \; \lim_{h\to 0}\frac{f(a+h)-f(a)}{h} \tag{1.5}$$

limiti varsa bu limite f fonksiyonunun a daki türevi denir. F 'nin a daki türevi $f'(a)$ ile gösterilir [41].

Örnek:

$$f(x) = 2x^2$$

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

$$= \lim_{h\to 0} \frac{2(x+h)^2 - 2x^2}{h}$$

$$= \lim_{h\to 0} \frac{2x^2 + 4xh + 2h^2 - 2x^2}{h}$$

$$= \lim_{h\to 0} \frac{4xh + 2h^2}{h} = \lim_{h\to 0} 4x + 2h = 4x$$

Türev bir takım kurallara sahiptir. Bu kurallar kullanılarak bir matematiksel ifadenin türevi daha kolay alınabilmektedir. Türev alma işlemi dört işlem için türev kuralları kullanıldığında farklı şekilde gerçekleştirilmektedir. Bu türev kuralları Tablo 3'te verilmiştir.

Tablo 3. Dört işlem için türev alma kuralları

Toplamanın Türevi	$(f+g)'(x) = f'(x) + g'(x)$
Farkın Türevi	$(f-g)'(x) = f'(x) - g'(x)$
Çarpımın Türevi	$(fg)'(x) = f'(x)g(x) + f(x)g'(x)$
Bölümün Türevi	$(f/g)'(x) = \frac{g(x)f'(x) - f(x)g'(x)}{(g(x))^2}$

Aynı şekilde üstel ve logaritmik fonksiyonların türevlerinin hesaplanması kolaylaştırmak için de kurallar geliştirilmiştir. Bu basit türev alma kurallarının bir kısmı Tablo 4'te verilmiştir.

Tablo 4. Üstel ve logaritmik fonksiyonların türevleri

$\frac{d}{dx}c^x = c^x \ln c$	$\frac{d}{dx}e^x = e^x$
$\frac{d}{dx}\log_c x = \frac{1}{x \ln c}$	$\frac{d}{dx}\ln x = \frac{1}{x}$
$\frac{d}{dx}x^x = x^x(1 + \ln x)$	

Trigonometrik fonksiyonların türevlerinin hesaplanmasını sağlayan türev kuralları ise sırasıyla Tablo 5'te verilmiştir.

Tablo 5. Trigonometrik fonksiyonların türevleri

$\frac{\mathrm{d}}{\mathrm{dx}}\sin x = \cos x$	$\frac{d}{dx}\cos x = -\sin x$
$(\tan x)' = \sec^2 x$	$\cot x = -\csc^2 x$
$\sec x = \sec x \tan x$	$\csc x = -\csc x \cot x$

1.3.6.2.2. L'Hopital Kuralları

- Kural I:

i. $\lim_{x \to a+} f(x) = 0 = \lim_{x \to a} g(x)$

ii. $\lim_{x \to a+} \frac{f(x)'}{g(x)'} = L$(L sonlu veya $\pm\infty$)

ise; $\lim_{x \to a} \frac{f(x)}{g(x)} = L$

- Kural II:

i. $\lim_{x \to a+} g(x) = \pm\infty$

ii. $\lim_{x \to a+} \frac{f(x)'}{g(x)'} = L$ (L sonlu veya $\pm\infty$)

ise; $\lim_{x \to a} \frac{f(x)}{g(x)} = L$

L'Hopital kurallarından da anlaşıldığı gibi, L'Hopital yöntemi [0/0] ve [∞/∞] belirsiz formlarına sahip ifadeler için doğrudan uygulanabilmektedir. Diğer belirsiz formlarına sahip ifadelere ise [0/0] ve [∞/∞] belirsiz formlarına dönüştürüldükten sonra L'Hopital yöntemi uygulanmaktadır. Tablo 6'da belirsiz formların [0/0] ve [∞/∞] belirsizliklerine dönüştürebilmek için gereken şartlar ve belirsizliklere dönüştürme sonucunda elde edilen ifadeler verilmiştir.

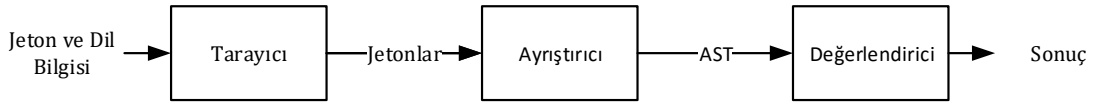
Tablo 6. Belirsiz form dönüşümleri

Belirsiz biçim	Şartlar	$0/0$ Dönüşümü	∞/∞ Dönüşümü
$0/0$	$\lim_{x\to c} f(x) = 0$ $\lim_{x\to c} g(x) = 0$	—	$\lim_{x\to c} \frac{f(x)}{g(x)} = \lim_{x\to c} \frac{1/g(x)}{1/f(x)}$
∞/∞	$\lim_{x\to c} f(x) = \infty$ $\lim_{x\to c} g(x) = \infty$	$\lim_{x\to c} \frac{f(x)}{g(x)} = \lim_{x\to c} \frac{1/g(x)}{1/f(x)}$	—
$0 * \infty$	$\lim_{x\to c} f(x) = 0$ $\lim_{x\to c} g(x) = \infty$	$\lim_{x\to c} f(x)g(x) = \lim_{x\to c} \frac{f(x)}{1/g(x)}$	$\lim_{x\to c} f(x)g(x) = \lim_{x\to c} \frac{g(x)}{1/f(x)}$
$\infty - \infty$	$\lim_{x\to c} f(x) = \infty$ $\lim_{x\to c} g(x) = \infty$	$\lim_{x\to c}(f(x) - g(x)) =$ $\lim_{x\to c} \frac{1/g(x) - 1/f(x)}{1/(f(x)g(x))}$	$\lim_{x\to c}(f(x) - g(x)) = \ln \lim_{x\to c} \frac{e^{f(x)}}{e^{g(x)}}$
0^0	$\lim_{x\to c} f(x) = 0^+$ $\lim_{x\to c} g(x) = 0$	$\lim_{x\to c} f(x)^{g(x)} = \exp \lim_{x\to c} \frac{g(x)}{1/\ln f(x)}$	$\lim_{x\to c} f(x)^{g(x)} = \exp \lim_{x\to c} \frac{\ln f(x)}{1/g(x)}$
1^∞	$\lim_{x\to c} f(x) = 1$ $\lim_{x\to c} g(x) = \infty$	$\lim_{x\to c} f(x)^{g(x)} = \exp \lim_{x\to c} \frac{\ln f(x)}{1/g(x)}$	$\lim_{x\to c} f(x)^{g(x)} = \exp \lim_{x\to c} \frac{g(x)}{1/\ln f(x)}$
∞^0	$\lim_{x\to c} f(x) = \infty$ $\lim_{x\to c} g(x) = 0$	$\lim_{x\to c} f(x)^{g(x)} = \exp \lim_{x\to c} \frac{g(x)}{1/g \ln f(x)}$	$\lim_{x\to c} f(x)^{g(x)} = \exp \lim_{x\to c} \frac{\ln f(x)}{1/g(x)}$

1.4. Matematiksel İfadelerin Ayrıştırılması

Matematiksel ifadeler cebir işlemlerinde yer alan rakamlar, fonksiyonlar, sabitler, değişkenler gibi matematiksel sembollerin tümünden oluşmaktadır. Sembollerden oluşan bir matematiksel ifadenin bilgisayar tarafından algılanıp, ayrıştırılması için derleyici adı verilen bir çeviriciye ihtiyaç vardır. Derleyiciler yüksek seviye dil ile yazılmış kaynak kodunu düşük seviye makine diline çeviren programlardır. Derleyicilere girdi verisi olarak

derleyicinin yapısına ve kurallarına uygun matematiksel ifadeler verildiğinde, çıktı olarak bu ifadenin makine dili karşılığı üretilmektedir. Bir matematiksel ifadenin ayrıştırma ve değerlendirilme süreci Şekil 1'deki gibidir.



Şekil 1. İfade ayrıştırma ve değerlendirme süreci

1.4.1. Sözcüksel (Lexical) Analiz

Derleme işleminin ilk aşaması girdiyi jetonlarına ayrıştırmaktır. Sözcüksel analiz girdi olarak bir karakter dizisi alarak buradaki satır başı, boşluk, sekme gibi karakterleri ve yorum satırlarını atlayarak işler ve bir kelime dizisi oluşturur. Bu aşamada tanımlanmamış bir kelime tespit edildiğinde lexical hatası rapor edilir. Bu aşama tarayıcı olarak da adlandırılır.

- Jeton :

Jeton bir dilin öğesi olan anlamlı karakter dizileridir. Her dil için tanımlı token kuralları vardır. Bu kurallar düzenli ifadelerle tanımlanır.

Örneğin tamsayılarla dört işlem yapan bir hesap makinesi dilinde aşağıdaki jetonlar bulunur.

Tablo 7. Dört işlem için sembol listesi

Toplam	+
Çıkarma	-
Çarpma	*
Bölme	/
Sol Parantez	(
Sağ Parantez	)
Sayı	[0-9]*

Ayrıştırma aşamasında kullanılan terimlerle ilgili açıklamalar aşağıdaki gibidir:

- Sembol:

Aritmetik ifade sembolleri +, -, *, /

- Alfabe:
  - Sonlu bir semboller kümesi;
  - {0,1} ikili alfabe kümesidir,
  - {0-9} sayı kümesidir,
  - {a-z, A-Z} bir alfabe kümesidir.
- Metin:

Sonlu bir alfabe dizisi bir metin kümesidir.

- Dil:

Metinler kümesidir.

- En Uzun Eşleşme Kuralı:

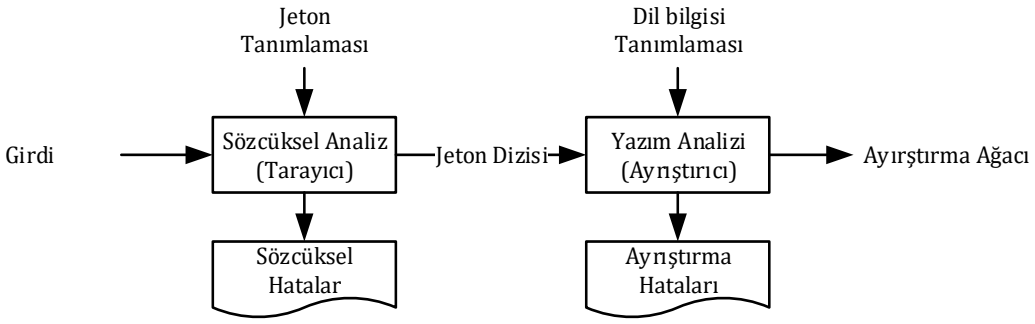
Sözcüksel analiz işleminde kaynak kod okunur, karakter karakter taranır ve bir boşluk, operatör sembölü veya özel simge ile karşılaştığında bir kelimeyi tamamladığına kadar verirlir.

- Tarayıcı Hatası:

Sözcüksel analiz hataları, kelimelerin, operatörlerin yazım yanlışlarını içerir.

1.4.2. Yazım(Sentaks) Analizi

Yazım analizinde bir önceki aşamadan(sözcüksel analiz) alınan jeton diziliminin dilin yazım yapısına uygunluğu kontrol edilir. Bu aşamanın çıktısı ayrıştırma ağacı(parse tree) denilen bir veri yapısıdır. Ayrıştırma ağacı girdi verinin dil bilgisi kurallarına göre şekillendirilmiş ve kolayca işlenebilir halidir. Bu aşamada herhangi bir jeton dil bilgisi kurallarına uymuyorsa yazım hatası raporlanır.



Şekil 2. Tarayıcı ve ayrıştırıcı Süreci

1.4.2.1. Ayrıştırma Ağaçları

Ayrıştırma işleminin sonucu ifadenin bütünsel anlamının korunduğu bir ağaç hiyerarşisinde sunulur.

Bir ayrıştırma ağacında:

- Her yaprak düğümünde terminaller bunulur.
- Her iç düğümde terminal olmayanlar bulunur.
- Sıraya dizilmesi orijinal giriş dizesini verir.

Matematiksel ifadeler için bir ayrıştırma ağacı birleşim ve operatör önceliklerini gösterir. Alt ağaçlar daha önce işleneceğinden dolayı alt ağaçtaki operatör üste göre daha yüksek önceliklidir.

1.4.2.2. Dillerin Hiyerarşisi

Bilgisayar bilimlerinin dil alanında yapılan çalışmalarında biçimsel dilleri sınıflandırmak için Chomsky Hiyerarşisi kullanılanılır. Literatürde Chomsky–Schützenberger hiyerarşisi olarak da geçmektedir. Bu yapıda diller dört tipe ayrılmıştır.

Dillerin bu yapıya göre sınıflandırılması Tablo 8'de verilmiştir.

Tablo 8. Chomsky hiyerarşisi

Tip	Dil	Makine	Karmaşıklık	Örnek
0	Özyinelemeli Diller	Turing Makinesi	Karar verilemez	Herhangi bir hesaplanabilir fonksiyon
1	Bağlama Duyarlı Diller	Doğrusal Bağlı Otomata	Üstel	$a^n b^n c^n$
2	Bağlam-Bağımsız Diller	Aşağı Sürüklemeli Otomata	Polinomial	$a^n b^n$
3	Düzenli Diller	Sonlu Otomata	Doğrusal	a*

1.4.2.3. Bağlam-bağımsız Diller

Düzenli ifadeler dilin jetonlarını ifade etmekte yeterli olmasına rağmen iç içe yapıları ifade edemezler. Tıpkı düzenli ifadeler gibi içerikten bağımsız diller de diziler kümesini ifade etmekle birlikte sembol dizilerinin yapısını da tanımlar.

Bir düzenli gramer şu bileşenden oluşur: G = (V, T, P, S)

- V: Sonlu bir terminal olmayan semboller kümesi.
- S ϵ V: Başlangıç sembolü.
- T: Terminal semboller kümesi.
- P: Yeniden yazma (üretim) kuralları kümesi.

1.4.2.4. Backus–Naur Biçimi

Backus–Naur Biçimi (BNF) bilgisayar bilimlerinde içerikten bağımsız diller için kullanılan bir notasyondur. BNF sadece yazım kuralları tanımlamak için değil anlam bilimsel kurallar içinde kullanılır. Derleyici üretim araçları YACC, LEX and JavaCC BNF notasyonunda dil tanımlaması kullanır.

Genişletilmiş (Extended) Backus–Naur Biçimi (EBNF) ve Artırılmış (Augmented) Backus–Naur Biçimi (ABNF) gibi çeşitleri vardır.

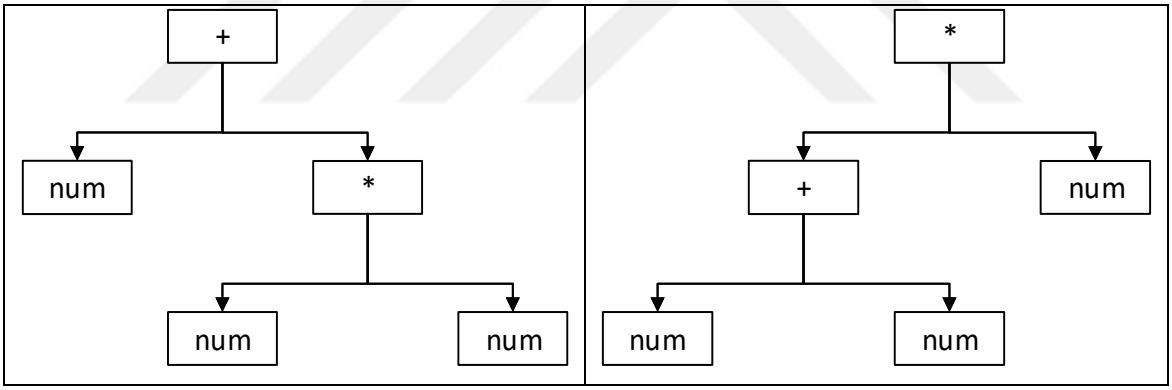
1.4.2.5. Belirsiz ve Belirli Dil Bilgisi

Bir katar için birden fazla soldan veya sağdan türetmeye imkân veren içerikten bağımsız dillere belirli (ambiguos) diller denir. Belirsiz (Unambiguous) diller ise her katar için sadece tek bir türetim garanti eden dillerdir. Örnek bir belirsiz dil tanımlaması Tablo 9'da verilmiştir.

Tablo 9. Belirsiz dil bilgisi örneği

```
E → E + E
E → E * E
E → num
```

Tablo 9'daki dil bilgisinin num+num*num girdisi için muhtemel iki farklı üretim yapabilir. Üretilen ağaç yapıları Şekil 3'deki gibidir.



Şekil 3. Belirsiz dilbilgisi ile üretilen ağaçlar

Örnek dil bilgisi belirsiz durumları giderecek şekilde yeniden yazılmadır. Bu bilgisinin sadece soldaki ağacı üretecek şekilde düzeltilmiş hali Tablo 10'daki gibidir.

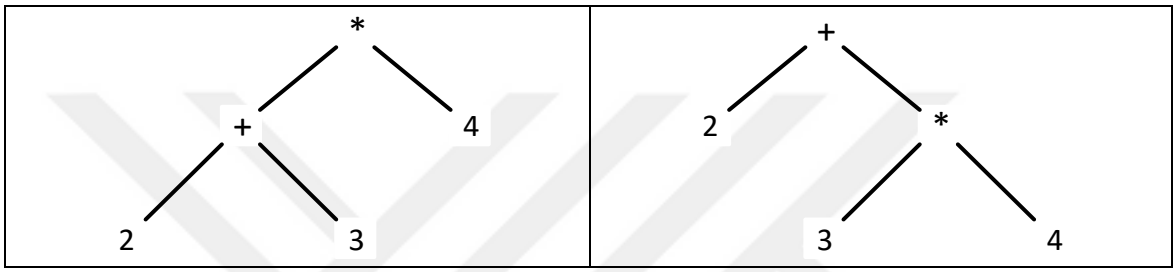
Tablo 10. Belirli dilbilgisi örneği

```
E → E + T
E → T
T → T * F
T → F
F → num
```

Belirsiz diller ayrıştırıcılar için sorun teşkil eder. Otomatik olarak belirsizliği tespit edip sonrasında da düzeltecek bir yöntem yoktur ve bu nedenle tek yöntem dil bilgisinin bu durumu ortadan kaldıracak şekilde yeniden yazılmasıdır.

1.4.2.6. Operatör Önceliği

Matematikte iki farklı operatör aynı işleneni paylaşıyorsa, operatör önceliği hangisinin işleneni işleme alacağını belirler.



Şekil 4. Örnek 2+3*4 ifadesi için operatör önceliği

1.4.2.7. Operatör Birleşimi

Bir işlenenin iki tarafında da operatör varsa işlenin hangi operatörü alacağı birleşme yönüyle belirlenir.

Örnek:
2^3^4 → 2^(3^4)
2^3^4 → (2^3)^4

1.4.2.8. First ve Follow Kümeleri

Öngörülü ayrıştırıcı geliştirmenin önemli bir aşaması da First ve Follow kümelerinin oluşturulmasıdır. Bu fonksiyonlar öngörülü ayrıştırıcı tablosunun giriş verilerini oluşturmamızı sağlar.

First(a), a'dan türetilen katarların muhtemel başlangıç terminalleri kümesidir.

FIRST(α) kümesi hesaplama algoritması:

- α bir başlangıç sembolü ise, FIRST(α) = { α }.
- α bir terminal olmayan ve α → Ɛ ise, FIRST(α) = { Ɛ }.
- α bir terminal olmayan ve α → $\boldsymbol{\gamma}$1 $\boldsymbol{\gamma}$2 $\boldsymbol{\gamma}$3…$\boldsymbol{\gamma}$n ve herhangi bir FIRST($\boldsymbol{\gamma}$) t yi barındırıyorsa, t FIRST(α) ya dâhildir.

Follow(a) ise hangi terminal sembollerinin a terminal olmayan sembolünün hemen ardından geleceğini beli terminal olmayan a sembolünün hemen ardından gelebilecek terminal sembolleri kümesi Follow(a) fonksiyonuyla gösterilir.

Follow kümesi hesaplama algoritması:

- α bir başlangıç sembolü ise, FOLLOW() = $
- α bir terminal olmayan ve α → AB, ise FIRST(B) FOLLOW(A) kümesine Ɛ hariç dâhildir.
- α bir terminal olmayan ve α → AB, ve B Ɛ, then FOLLOW(A) kümesi FOLLOW(α) kümesine dâhildir.

1.4.2.9. Ayrıştırma Algoritmaları

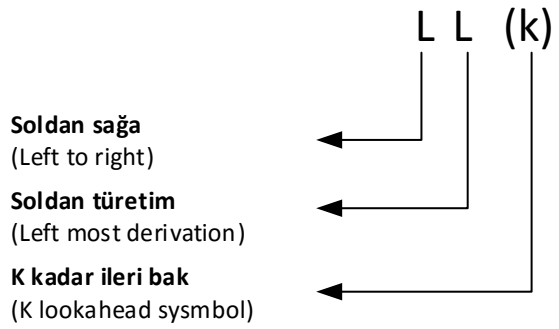
Ayrıştırma algoritmaları ağacı oluşturma şekilleri açısından olarak iki kategoride incelenir:

Yukarıdan-aşağıya ayrıştırma algoritmasında ayrıştırma ağacı kökden başlayarak aşağıya doğru oluşturulur. En yaygın örneği LL ayrıştırma algoritmalarıdır.

Aşağıdan-yukarı ayrıştırmasında algoritmasında ise ağaç yapraklardan başlanarak yukarı doğru oluşturulur. Bu türün en yaygın üyesi LR ayrıştırma algoritmalarıdır.

1.4.2.9.1. LL Ayrıştırma

LL grammer kullanan parserlardır. LL grammer bazı kısıtlamaları, kuralları olan bir tür bağlamdan bağımsız grammer tipidir.



Şekil 5. LL ayrıştırma

LL(1) yukarıdan aşağıya, hedefe odaklı ve öngörülü bir tekniktir. Bir LL grammerde olmaması gereken özellikler:

- Belirsiz dilbilgisi
- Soldan yinelemeli
- Soldan üretimli

1.4.2.9.1.1 Soldan yineleme

FIRST(T) kümesindeki herhangi bir sembol aynı zamanda FIRST (E+T) kümesinde de varsa bu durum soldan yineleme sorunu oluşturur. Bu sorun E sembolünün sağındaki ilk sembolün yine E olmasından kaynaklanır.

$E \rightarrow E + T$

$E \rightarrow T$

Soldan yineleme durumunu gidermek için dil bilgisinin sağdan yinelemeli olacak şekilde yeniden yazılması gerekir.

$E \rightarrow T\ E'$

$E' \rightarrow + T\ E'$

$E' \rightarrow$

1.4.2.9.1.2 Sol Çarpanlama

Ayrıştırıcılar sağdan sola doğru okuma yaparak çalışmaktadır. Jeton listesinden okurken bir taraftanda hangi dilbilgisi kuralına göre işlem yapacağı kararını alması gerekir. En az iki kuralın aynı sembol veya sembollerle başladığı durumlarda hangi kural üzerinden üretime devam edileceği belirsiz bir durum oluşturur.

S → a b
S → a c

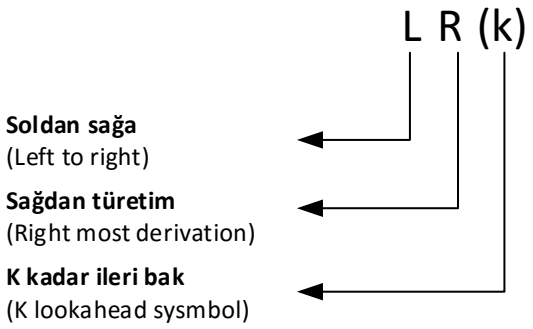
Sol çarpanlama dil bilgisinin yukarıdan aşağıya ayrıştırıcılar için kullanılabilir şekle çevirmektir. Burada yapılan işlem her ortak örnek içim bir üretim yapılır ve geri kalan türetim yeni üretimlere ilave edilir.

S → a S’
S’ → b
S’ → c

Sol çarpanlama durumunu engelleyen yeni dil bilgisinde ayrıştırıcı aynı sembolle başlayan kurallar arasına seçim yapma problemiyle karşılaşmayacak.

1.4.2.9.2. LR Ayrıştırma

LL(k) ayrıştırma yöntemlerinde ilk k tane jetona bakarark hangi üretimin yapılacağının öngörülmesi gerekmesi bir dezavantajdır. Buna karşılık LR(k) ise karar verme sağ taraftaki tüm jetonları görene kadar ertelemek mümkündür. Bu sayede öngörüde bulunması gerekmez.



Şekil 6. LR ayrıştırma

1.4.3. Anlambilimsel (Semantik) Analiz

Bir cümle yazım olarak doğru fakat anlam olarak hatalı olabilir. Dil kurallarının bazıları semantik analiz aşamasında kontrol edilir. Bu aşamada öğelerin kullanım şekillerine bakar ve ifadelerin doğru şekilde oluşturulup oluşturulmadığını kontrol eder. Örneğin bir sayı veri tipi ile bir karakter dizisi tipi kıyaslaması yapılamaz. Ayrıca bu aşamada semantik analiz bazı durumlarda söz dizim analizi ile birlikte yapılabilir veya hiç yapılmayabilir.

1.5. Ayrıştırıcı Üreteçleri

Ayrıştırıcı tasarımı bilgisayar bilimlerinin programlama dili geliştirilen uç alanlarından biridir. Ayrıştırıcı üretimi için kullanılan lisanlı veya açık kaynak kodlu birçok alternatif yazılım vardır. Bunlardan biri olan yacc oluşturulan bir gramerden o dil için ayrıştırıcı üretirken, Lex ise tanımlamalardan sözcüksel analiz programı üretir. Yacc ve Lex C programlama dili tabanlı yazılımlardır. JavaCC ve SableCC gibi Java programlama dili tabanlı ayrıştırıcı üretim araçlarıda vardır. Bu iki araçta ücretsiz kullanılabilir ve internette ve kitaplarda bunlarla ilgili yeterli kaynak bulunmaktadır.

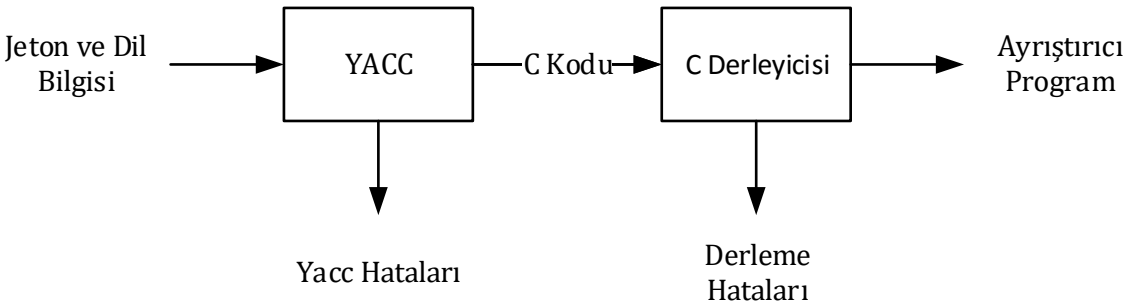
Ayrıştırıcı üreteçleri yukarıdan aşağıya ve aşağıdan yukarı olmak üzere iki tiptir. Ayrıştırıcı üreteceleri arasında gerek geliştirildikleri dil, kullandıkları algoritma gerekse dil bilgisi tipleri bakımından farklılık gösterir.

Tablo 11. Tarayıcı ve ayrıştırıcı üreteçleri

Regular Dil				
Lexer üreteci	**Algoritma**	**Dil**		**Lisans**
Lex	DFA	C		Proprietary, CDDL
Flex	DFA	C & C++		BSD
JFlex	DFA	Java		GNU GPL
Deterministik Bağlamdan Bağımsız Dil				
Parser üreteci	**Algoritma**	**Grammer**	**Dil**	**Lisans**
Yacc	LALR(1)	YACC	C	CPL & CDDL
Bison	LALR(1), LR(1), IELR(1), GLR	YACC	C, C++	GNU GPL
JavaCC	LL(k)	EBNF	Java, C++, JavaScript (via GWT compiler)[1]	BSD
SableCC	LALR(1)	EBNF	C, C++, C#, Java, OCaml, Python	GNU LGPL

1.5.1. Yacc

En yaygın ve popüler ayrıştırıcı üreteçlerinden biri Yacc'dır. Yacc LALR parserları üretir ve çıkış olarak C programla dilinden bir kod üretir. Lex ya da Flex kullanır.



Şekil 7. Yacc ile ayrıştırıcı üretimi

Yacc ile tam uyumlu, açık kaynak kodlu örneği Bison'dur. Yacc için yazılmış tüm dil bilgisi tanımlama dosyaları Bison'da sorunsuz çalışır.

1.5.2. JavaCC

JavaCC düzenli ifadelerden sözcüksel analiz ve bağlamdan bağımsız gramerlerden ise ayrıştırıcı üreten java ile geliştirilmiş bir araçtır.

JavaCC kullanımın sağladığı avantajlar:

- Ayrıştırıcı/derleyici üretiminde zamandan tasarruf etme imkânı sağlar.
- Standart bir kodlama sunar.
- Hatasız geliştirme imkânı sunar.
- Java dili ile nesne tabanlı geliştirme imkânı sağlar.
- Regex ve grammer tanımlaması doğru yapılmışsa hatasız çalışan bir ayrıştırıcı üretir.

JavaCC ".jj" uzantılı bir yapılandırma dosyası kullanır. Bu dosyası seçeneklerin ayarlanması ile başlar. Bu seçenekler arasında öngörülü üretim yaparken bakılacak jeton sayısı, hata ayıklama modunu aktif/pasif etme, oluşturulacak dosyaların hedef klasörünü belirleme gibi maddeler vardır.

Ardından üretilecek ayrıştırıcının ana sınıfının gövdesinin tanımlaması yapılır. Bu *PARSER_BEGIN* ve *PARSER_END* etiketleri arasında ana ayrıştırıcı sınıfının tanımlaması ve gövdesi yapılabilir. Bu alana eklenecek her türlü kod JavaCC tarafından oluşturulacak sınıfa aynen tanışır.

Üçüncü aşama olarak sözcüksel analiz için gereken atlanacak karakterler ve jeton listesi düzenli ifadeler kullanılarak tanımlanır. Jeton listesi *TOKEN* etiketi altında gerekirse düzenli ifadeler de kullanılarak tanımlanır. Atlanacak karakterler *SKIP* etiketi ile tanımlanır ve genellikle boşluk ve satır sonu karakterlerdir. Bunlara uygulamaya göre başka karakterlerde eklenebilir.

Tablo 12'de JavaCC yapılandırma dosyası için bir örnek verilmiştir.

Tablo 12. JavaCC yapılandırma dosyası

```
OPTIONS{
LOOKAHEAD = 1;
...
}
```

```
PARSER_BEGIN(parser_name)
. . .
class parser_name . . . {
  . . .
}
. . .
PARSER_END(parser_name)
```

```
SKIP  :{ " " | "\t" | "\n" }
TOKEN :
{ < NUM  : (["0"-"9"])* >
| < PLUS : "+" >
| < MINUS: "-" >
...
}
```

Yapılandırma dosyası son olarak dil bilgisinin tanımlanmalarının eklenmesi ile tamamlanmış olur. JavaCC dilbilgisi tanımlanmasında genişletimiş BNF kullanır.

1.5.2.1. EBNF

Genişletilmiş BNF (EBNF) bazı düzenli ifadelerininde kullanılabildiği bir BNF türüdür. Bu ek kullanımlar:

- "*" : 0 veya daha fazla örneği anlamına gelir
- "+" : 1 veya daha fazla örneği anlamına gelir
- "?" : 0 veya 1 örneği anlamına gelir
- Gruplama için parantez kullanılabilir.

Tablo 13. Sayı ifadesi için BNF türünden dil bilgisi

```
<expr>   ::= '-' <num> | <num>
<num>    ::= <digits> | <digits> '.' <digits>
<digits> ::= <digit> | <digit> <digits>
<digit>  ::= '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
```

Tablo 13'deki BNF türünden dil bilgisinin EBNF türünde ifade edilmesi Tablo 14'de de görüleceği üzere daha kısadır.

Tablo 14. Sayı ifadesi için EBNF dil bilgisi

```
<expr>  := '-'? <digit>+ ('.' <digit>+)?
<digit> := '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
```

1.5.2.2. AST

Soyut sözdizim ağacı, iç düğümlerinde operatörler, yaprak düğümlerinde ise düğüm operatörlerin işlenen bulunan sonlu ve yönlendirilmiş bir ağaç yapısıdır. JavaCC ayrıştırıcı aşaması sonunda çıkışı soyut nesne ağacıdır. Modülerliği artırmak için söz dizim ve semantik konularını ayrırmak faydalıdır. JavaCC ayrıştırıcı aşaması sonunda çıkışı bir sonraki aşamalarda kullanılacak bir veri yapısı olan ayrıştırma ağacı üretir.

AST ayrıştırıcı ve sonraki derleyici aşamaları arasıda temiz bir ara yüz sağlar. Bu ağaç kaynak programının ifade yapısını taşır. Bütün ayrıştırma aşaması semantik yorumlama olmadan tamamlanmıştır.

Sadece toplama, çıkarma, bölme ve çarpma işlemleri olan bir matematiksel ifade için yazılacak dil bilgisi aşağıdaki gibidir.

Tablo 15. Dört işlem için örnek dil bilgisi

```
E → E + E
E → E − E
E → E * E
E → E / E
E → num
```

Bu dil bilgisine karşılık oluşturulacak AST sınıfları Tablo 16'daki gibidir.

Tablo 16. Dört işlem için AST sınıfları

```
public abstract class Exp {}
public class Plus extends Exp {
     public Exp e1,e2;
     public Plus(Exp a1, Exp a2) { e1=a1; e2=a2; }
}
public class Minus extends Exp {
     ...
}
public class Times extends Exp {
     ...
}
public class Divide extends Exp {
     ...
}
public class Num extends Exp {
     public int n;
     public Num(int n) { this.n = n; }
}
```

Yukarıdaki sınıflardan AST üretecek ayrıştırıcı dil bilgisi Tablo 17'deki gibidir.

Tablo 17. Dört işlem için dil bilgisi

```
Exp Start() :
{ Exp e; }
{ e=E() { return e; }
}
Exp E() :
{ Exp e1,e2; }
{ e1=T() ( "+" e2=T() { e1 = new Plus(e1,e2); }
         | "-" e2=T() { e1 = new Minus(e1,e2); })*
{ return e1; }
}
Exp T() :
{ Exp e1,e2; }
{ e1=F() ( "*" e2=F() { e1 = new Times(e1,e2); }
         | "/" e2=F() { e1 = new Divide(e1,e2); })*
{ return e1; }
}
Exp F() :
{ Token t; }
{ t=<NUM> { return new Num(Integer.parseInt(t.image)); }
}
```

$3 + 4 * 5$ girdisi için tanımlanan dosya ile oluşturulmuş ayrıştırıcının üreteceği nesne ağacı şu şekildedir:

new Times(new Plus(new Num(3),new Num(4)), new Num(5))

1.5.2.3. Ziyaretçi Tasarım Şablonu

Ziyaretçi tasarım şablonu Erich Gamma tarafından tanıtılan tasarım desenlerinden biridir [42]. İki amaca hizmet eder.

1) Ağaç üzerinde dolaşma işlemi ile düğümde yapılacak eylemleri birbirinden ayırır.
2) Her aşamanın bütün eylemlerini aynı yapı altına toplar.

Ziyaretçi tasarım şablonunda öncelikle bir ziyaretçi arayüzü tanımlanır. Her bir nesneye karşılık bir visit methodu içerir. Bu arayüz sınıfında sadece metotlar açıklanır ve metotların gövdeleri boştur.

Örnek bir ziyaretçi arayüz sınıfı Tablo 18'deki gibidir.

Tablo 18. Visitor arayüzü

```
public interface IVisitor {
     public int visit(Plus e);
     public int visit(Minus e);
     public int visit(Times e);
     public int visit(Divide e);
     public int visit(Num e);
}
```

Tanımlanan arayüz yapısından bir ziyaretçi sınıfı oluşturulur. Bu sınıf arayüzdeki bütün methodların gövdeleri için gerekli kodları barındırırır. Bir arayüzden birden fazla ziyaretçi sınıfı için kullanılabilir.

Tablo 18'deki ziyaretçi arayüzünü gerçekleyen bir ziyaretçi sınıfı Tablo 19'daki gibidir.

Tablo 19. Visitor sınıfı

```
public class Visitor implements IVisitor {
     public int visit(Plus e) {
          return e.e1.accept(this)+e.e2.accept(this);
     }
     public int visit(Minus e) {
          return e.e1.accept(this)-e.e2.accept(this);
     }
     public int visit(Times e) {
          return e.e1.accept(this)*e.e2.accept(this);
     }
     public int visit(Divide e) {
          return e.e1.accept(this)/e.e2.accept(this);
     }
     public int visit(Num e) {
          return e.n;
     }
}
```

Ziyaretçi tasarım şablonu diğer yöntemlere göre kaynak kodunun daha düzenli, okunabilir olmasına imkân tanır. Ayrıca koda bir ekleme veya değişiklik yapılmasını geliştirici açısından kolaylaştırır. Ziyaretçi her AST sınıfı için bir visit methoduna sahip bir yorumlayıcı nesnedir. Bununla birlikte bu yapıda her AST sınıfının, görevi nesneyi uygun visit methoduna göndermek olan bir accept methodu olmalıdır. Bu şekilde kontrol visitor ile AST sınıfı arasında gider gelir.

Ziyaretçi tarafından çağrılan düğümün accept methodu düğümün sınıf tipiyle uygun visit methodunu çağırır. Bu nesne ağacının sonuna kadar devam eder.

Özetle ziyaretçi tasarım şablonu mevcut yapıya dokunmadan yeni bir değerlendirici eklenmesine imkân sağlar.

1.6. Kullanılan Diğer Teknolojiler

Sistemin geliştirilme sürecinde JavaCC ayrıştırma üreteci dışında birçok ürün ve teknolojiden faydalanılmıştır. Bunlar Apache Maven, Junit, Jsp, Tomcat ve Mathjax'dır.

1.6.1. Maven

Apache Maven bir yazılım proje yönetimi ve anlama aracıdır. Bir proje nesne modeli (POM) kavramına dayanarak, maven bir projenin inşa, raporlama ve dokümantasyon süreçlerini yönetebilir [43].

Maven; proje oluşturmak için standart bir yol, projenin nelerden oluştuğunun net bir tanımı, proje bilgilerini yayımlamak için kolay bir yol ve çeşitli projeler arasında JAR paylaşmak için bir yol sunar.

Maven'ın öncelikli hedefi geliştiriciye kısa sürede bir geliştirme çabasının tam durumunu anlamasına imkân sağlamaktır. Bu hedefe ulaşmak için Maven'ın başa çıkması gereken çeşitli ilgi alanları vardır:

- İnşa sürecini kolaylaştırma,
- Düzgün bir yapı sistemi sağlanması,
- Kaliteli proje bilgisi sağlanması,
- En iyi geliştirme için yönergeler sağlaması,
- Yeni özelliklere şeffaf geçiş sağlaması

1.6.2. Junit

JUnit Java programlama dilinde test birim testleri için kullanılan bir çatıdır. JUnit test güdümlü program geliştirmede önemli bir öğedir ve xUnit olarak bilinen birim test çatıları ailesinin bir öğesidir [44].

JUnit derleme zamanında bir JAR olarak bağlanır ve bu çatı önceleri junit.framework paketi altındaki iken JUnit 4 ve sonrası org.JUnit paketi altındadır.

1.6.3. Jsp

Java Server Pages(JSP) teknolojisi kolayca dinamik ve statik web içerik üretmeye imkân sağlar. JSP teknolojisi Java Servlet'in bütün dinamik imkânlarına sahipken ayrıca ek olarak statik içerik için doğal bir yaklaşım sunar [45].

JSP'nin özellikleri:

- JSP sayfalarını geliştirmek için bir dildir.
- Sunucu tarafı nesneleri erişmek için bir ifade dilidir
- JSP uzantıları tanımlamak için mekanizmalar sunar.

1.6.4. Tomcat

Tomcat, Apache vakfı tarafından geliştirilen açık kaynak kodlu Apache vakfı tarafından geliştirilen bir web sunucusudur. Tomcat platform bağımsız bir uygulama olup, Java Servlet, JSP, Java EL ve WebSocket gibi teknolojileri desteklemektedir. Ayrıca Java kodunun çalıştırılabileceği saf Java http web sunucusu ortamı sağlar.

1.6.5. Mathjax

MathJax matematik görüntüleme için açık kaynak kodlu bir Javascript platformudur. MatJax'ın bazı özellikleri:

- Tüm tarayıcılarda yüksek kaliteli matematiksel görüntü sağlar.
- Tarayıcı için özel bi kurulum gerektirmemektedir.
- HTML kaynağı LaTeX, MathML ve doğrudan diğer denklem işaretleme için destek sunar.
- Web uygulamalarına kolay entegrasyon için zengin bir API barındırır.
- Erişilebilirlik desteği, kopyala ve yapıştır ve diğer zengin işlevsellik sunar.
- Diğer uygulamalarla birlikte çalışabilmektedir.

2. YAPILAN ÇALIŞMALAR, BULGULAR VE İRDELEME

2.1. Giriş

Tez kapsamında yapılan çalışmalar “belirsiz limit ifadelerin adım adım çözümü” ve “otomatik belirsiz limit ifadeleri üretimi” olmak üzere iki alt çalışmadan oluşmaktadır.

Birinci bölümde limit ifadelerinin dil bilgisi incelenmiş ve dil bilgisi çıkarımı yapılmıştır ve bu dil bilgisi ile limit ifadelerinin ayrıştırılması gerçekleştirilmiştir. Ayrıca limit ifadelerinin değerlendirilmesinde ise özellikle belirsiz limit ifadelerinin L’Hopital kuralı ile çözümü, sembolik türev alma, sembolik sadeleştirme, trigonometrik fonksiyonlar gibi konular üzerinde durulmuştur. L’Hopital kuralı ile belirsiz limit ifadelerinin çözümü üzerine bir sistem geliştirilmiştir.

İkinci bölümde ise rastgele ifade üretimi üzrinde durulmuş, mevcut yöntemler ele alınmıştır. Bu bölümde otomatik belirsiz limit ifadeleri üretme konusu incelenmiş, rastgele AST üretimi tabına dayanan bir yöntem sunulmuş ve buna yönelik bir sistem tasarlanıp gerçeklenmiştir.

2.2. Belirsiz Limit Sorularının Adım Adım Çözümü

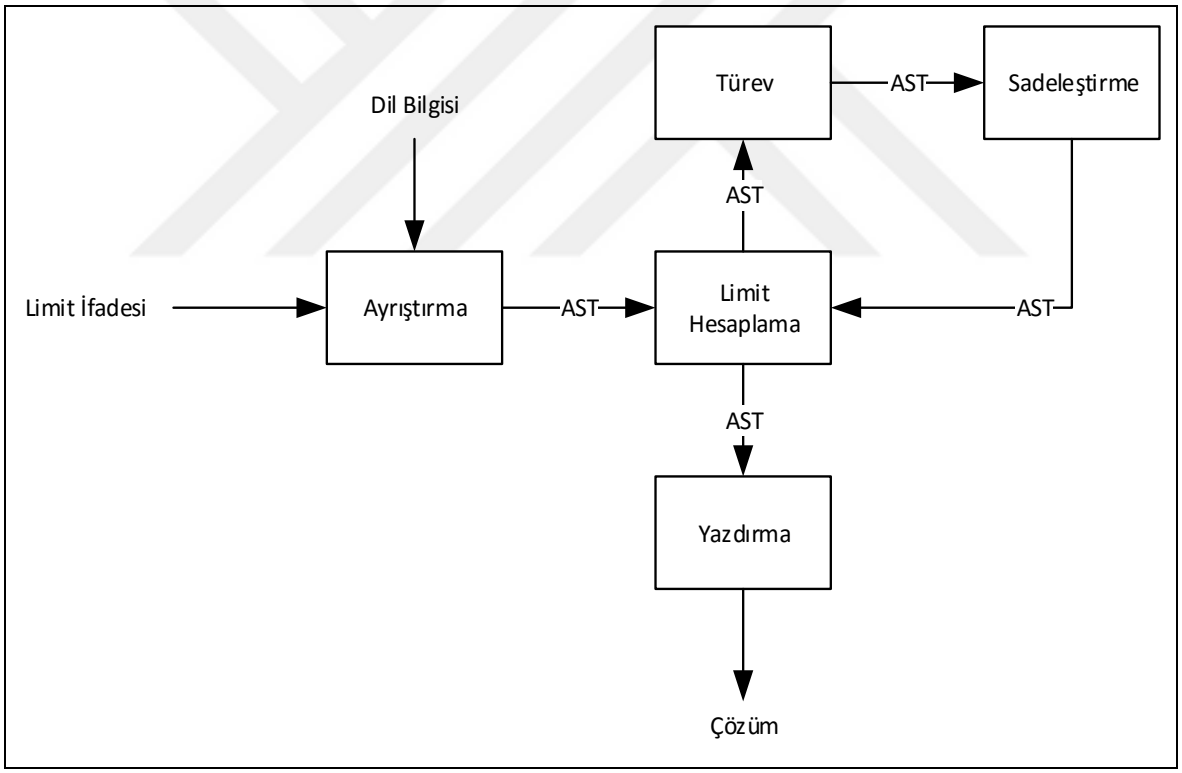
Bu bölümde limit problemlerinin simgesel hesaplama yöntemiyle adım adım çözülebilmesi için tasarlanan sistemin sunumu yapılacaktır. Bu kapsamda öncelikle belirsiz limit problemleri ile ilgili kaynaklar incelenerek çözüm yolları araştırılmıştır. İncelemeler sonucunda genişletilmiş dil bilgisi tanımlaması oluşturulmuştur. Simgesel hesaplama yapabilmek için öncelikle matematiksel ifadenin ayrıştırılması gerekmektedir. Bu işlem için bir ayrıştırıcı üreteci olan JavaCC seçilmiş ve oluşturulan dil bilgisi JavaCC'de tanımlaması yapılarak limit ifadeleri için bir ayrıştırıcı geliştirilmiştir.

Ayrıştırma işlemi sonucunda matematiksel ifadenin Java nesneleriyle temsil edildiği hiyerarşik yapıya sahip bir nesne ağacı oluşturulur. Bundan sonraki aşamada ise değerlendirme süreci başlatılır ve ziyaretçi tasarım şablonunu oluşturan değerlendirici metotlar ile nesne ağacı yorumlanır. Bu aşamada bir belirsizlik tespit edilirse, L’Hopital teoremine göre türev işlemi gerçekleştirilir. Türev işlemi sonucunda varsa sadeleştirme yapılır.

Bir limit ifadesinin çözüm adımlarını gösterebilmek için her bir işlem adımı sonrası nesne ağacında oluşan ifade yazdırılır. İfade çözümleyicinin ana bileşenleri aşağıdaki gibi listelenebilir:

- Simge listesi oluşturma
- Dil bilgisi (gramer) kurallarını belirleme
- AST için gereken sınıfları oluşturma
- Değerlendirici ziyaretçi (Visitor) sınıfları oluşturma
- Yazdırıcı sınıfları oluşturma

Genel olarak baktığımızda ise çalışmada gerçeklenen sistemin genel yapısı Şekil 8'de gösterilmiştir.



Şekil 8. Limit ifadeleri için çözümleyici ana bileşenleri

2.2.1. Limit Grameri

Simge listesi ve dil bilgisi kurallarının belirlenebilmesi için öncelikle limit ifadelerinin genel yapısını incelemek gerekmektedir. Limit fonksiyonunun genel matematiksel tanımı aşağıdaki gibidir [39]:

$$\lim_{x \to a} f(x) = L \tag{2.1}$$

Bu ifadede x, a'ya yaklaşıyorken $f(x)$'in limitinin L olacağı söylenir. L sonlu olabilir ya da $-\infty$, $+\infty$ değerlerini alabilir. Limit fonksiyonunun çözümü $f(x)$ fonksiyonunda x yerine a değeri koyularak yapılmaktadır. Çalışmada limit fonksiyonunun genel biçimi metin formatında "lim (x->a) f(x)" şeklinde temsil edilmiştir.

Limit ifadelerinin grameri incelendiğinde, sabit bir 'lim' sözcüğü ile başlar, 'a' yerine bir sayısal değer (genellikle bir tamsayı) yazılır ve '$f(x)$' için de bir matematiksel ifade girilir. Bu nedenle bir limit ifadesinin bütün bileşenlerini temsil etmek üzere bir simgeler (token) listesi oluşturulmuş olup, Tablo 21'de gösterildiği gibi, bir ".jj" uzantılı dosya içerisinde JavaCC formatında tanımlanmıştır.

2.2.2. JavaCC Dosyası

JavaCC dosyası ana ayrıştırıcı sınıfı, simge listesi, seçenekler ve dil bilgisi kuralları gibi birkaç alt bölümün birleşiminden oluşur. Bu amaçla ana sınıf com.mcaydogdu.parser paketi içerisinde *Limit* olarak adlandırılmıştır ve Tablo 20'de örneği gösterilmiştir. Belirlenen simge listesi *Token* etiketi altında Tablo 21'deki gibi tanımlanmıştır.

Tablo 20. JavaCC .jj dosyası tanımlaması

```
options {
DEBUG_PARSER=false; LOOKHEAD=1;
}
PARSER_BEGIN(Limit)
package com.mcaydogdu.parser;
  public class Limit {
...
  }
PARSER_END(Limit)
SKIP: { " " | "\t" | "\r" }
```

Tablo 21. Limit ifadeleri için JavaCC jeton tanımlaması

```
TOKEN: {
  < X: "x" >
| < NUM: (["0"-"9"])+ >
| < LIMIT: "lim" >
| < LPR: "(" >
| < RPR: ")" >
| < RARW: "->" >
| < PLUS: "+" >
| < MINUS: "-" >
| < TIMES: "*" >
| < DIVIDE: "/" >
| < POWER: "^" >
| < SIN: "sin" >
| < COS: "cos" >
| < TAN: "tan" >
| < COTAN: "cotan" >
| < SEC: "sec" >
| < COSEC: "cosec" >
| < LN: "ln" >
| < EXP: "exp" >
| < SQRT: "sqrt" >
| < INF: "inf" >
| < PI: "pi" >
}
```

Tablo 21'de yapılan tanımlamalara göre örnek bir "$lim\ (x \to 2)(x+2)/(x-1)$" ifadesi için Javacc'nin oluşturacağı simge dizisi aşağıdaki gibi olacaktır:

<LIMIT> <LPR> <X> <RARW><NUM><RPR> <LPR><X><PLUS><NUM><RPR> <DIVIDE> <LPR><X><MINUS><NUM><RPR>

Simge listesi çıkarımı yapıldıktan sonra genel limit ifadesini kapsayacak dil bilgisi çıkarımı yapılmıştır. Limit ifadelerinin dil bilgisi Tablo 22'deki gibi yazılabilir. JavaCC LL(1) dil bilgisi kullandığı için tanımlama bu türden yapılmıştır.

Tablo 22. Limit ifadeleri için LL(1) dil bilgisi

```
G={Σ, T, V, P, S}
V={Limit, Expr, Elem, Term, Unary, Power, Func, Num, A}⊆ Σ
T={x, pi, inf, Sin, Cos, Tan, Log, Exp, Sqrt, Ln, ( , ), +, -, *,
/, ^}⊆ Σ
Σ=T ∪ V
S={Limit}
Productions
<Limit> → "lim (x->" <A> ")" <Expr>
<A>     → ("-")? "pi" | "inf" | <Num>
<Expr>  → ("+"|"-")? <term> [("+"|"-") <term> ]*
<Term>  → <power> [("*"|"/") <power>]*
<Power> → <element> ("^"<power>)?
<Elem>  → <func> "(" <Expr> ")" | <number> | "x"
<Func>  → "Sin" | "Cos" | "Tan" | "CoTan" | "Sec"
<Func>  → "CoSec" | "Log" | "Exp" | "Sqrt" | "Ln"
<Num>   → "-"? ["0"-"9"] + ("."["0"-"9"]+)?
```

Dil bilgisi tanımlanırken operatör öncelikleri, birleşme yönleri, terimlerin işaretleri gibi özellikler göz önünde bulundurulmuştur. Operatörler arasında üst alma işlemi en yüksek öncelikte iken toplama ve çıkarma operatörleri en düşük önceliğe sahiptir. Limit ifadelerinden çıkarılan dil bilgisinin EBNF formatındaki tanımlaması Tablo 23'de verilmiştir.

Tablo 23. Limit ifadeleri için JavaCC dil bilgisi tanımlaması

```
Lim Prog():
{ Exp e; Token t; int s=1; }
{
  <LIMIT> <LPR> <X> <RARW> (<PLUS>|<MINUS> {s=-1;} )?
(t=<NUM>|t=<INF>|t=<PI>) <RPR> e=E() (<EOF> | "\n") { return new
Lim(...); }
}
Exp E():
{ Exp e1, e2; int s=1;}
{ e1=T(s) ( <PLUS>  e2=T(s) { e1 = new Plus(e1, e2); }
          | <MINUS> e2=T(s) { e1 = new Minus(e1, e2); } )*
{ return e1; }
}
Exp T(int s):
{ Exp e1, e2; }
{ e1=K(s) ( <TIMES>  e2=K(s) { e1 = new Times(e1, e2); }
          | <DIVIDE> e2=K(s) { e1 = new Divide(e1, e2); } )*
{ return e1; }
}
Exp K(int s):
{ Exp e1, e2; }
{ e1=F(s) ( <POWER>  e2=F(s) { e1 = new Power(e1, e2); } )*
{ return e1; }
}
Exp F(int s):
{ Exp e=null; Exp a = null;Token t; }
{ (<PLUS>|<MINUS> {s=-1;})? (
  <X> {if(s==1) {return new X();} else {return new Times(new Num(-
1),new X());} }
  | <LPR> e=E() <RPR> { return e; }
  | <LN> <LPR> e=E() <RPR> { return new Ln(e); }
  | <SIN> <LPR> e=E() <RPR> { return new Sin(e); }
  ... )
}
```

2.2.3. AST Tanımlamaları

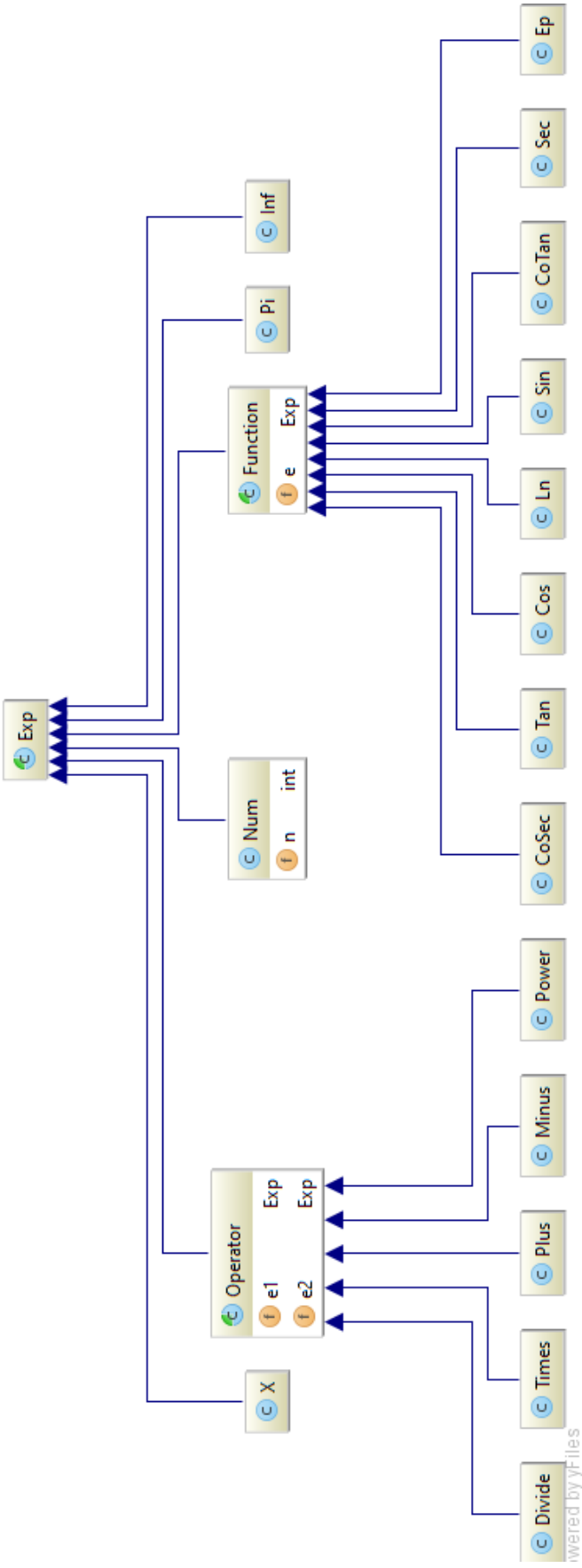
AST sınıfları en genel matematiksel ifadeyi temsil eden Exp sınıfından türetilerek, dil bilgisinde bulunabilen bütün alt ifadelerin simgesel bileşenleriyle uyumlu olarak tanımlanmıştır. Örneğin, işlem operatörlerini (toplama, çıkarma vb.) içeren alt ifadeler için oluşturulan her bir sınıfa Exp türünden iki nesne referansı eklenirken, sadece bir sayıdan oluşan alt ifadeler için oluşturulan Num sınıfına bir double verisi eklenmiştir.

Çalışmada oluşturulan AST sınıflarından bazıları Tablo 24'de verilmiştir.

Tablo 24. Limit ifadeleri için oluşturulan AST sınıfları

```
abstract class Exp {
    public abstract double accept(ILimitVisitor v);
}
…
class Plus extends Exp {
    Exp e1, e2;
    public Plus(Exp x, Exp y) { e1 = x; e2 = y; }
    public double accept(ILimitVisitor v)
    { return v.visit(this); }
}
…
class Num extends Exp {
    double n;
    public Num(double x) { n = x; }
    public double accept(ILimitVisitor v)
    { return v.visit(this); }
}
class X extends Exp {
    public double accept(ILimitVisitor v)
    { return v.visit(this); }
}
```

Oluşturulan AST sınıfları kullanım kolaylığı sağlaması açısından fonksiyonlar Function, işlem operatörleri ise Operator sınıflarından türetilmiştir. Bu yapıyı gösteren sınıf diyagramı Şekil 9'de verilmiştir.

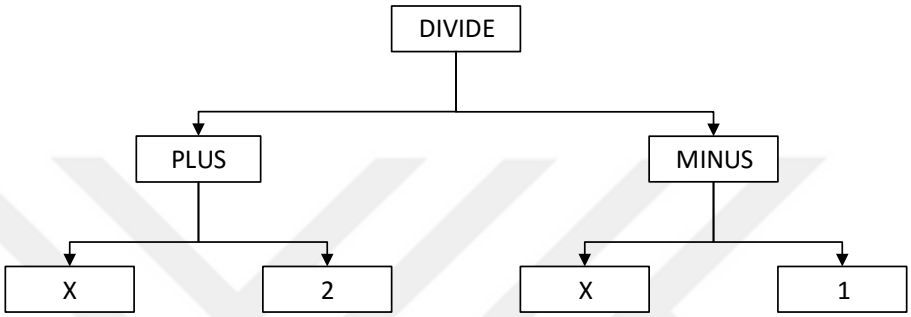


Şekil 9. Oluşturulan AST sınıf diyagramı

Bir örnekle açıklamak gerekirse;

$$\lim_{x \to 2} \frac{x+2}{x-1}$$

limit ifadesi için JavaCC ile oluşturulacak AST ağacı Şekil 10'de gösterilmiştir.



Şekil 10. Örnek limit ifadesi için oluşturulan AST

Şekil 10'da gösterilen nesne ağacının Java dili ifadeleri ile temsili ise aşağıdaki gibi yapılmaktadır:

Exp tree = new Divide(new Plus(new Num(2), new X()), new Minus(new X(0), new Num(1)))

2.2.4. Limit Değerlendirme

Çalışmada yukarıda bahsedilen özelliklerinden dolayı ağaç değerlendirme sürecinde ziyaretçi tasarım şablonu tercih edilmiştir. Fakat ağaç değerlendirme süreci için literatürde kullanılan başka yöntemler de mevcuttur [46].

Ayrıştırılmış bir limit ifadesini değerlendirmek için geliştirilen ziyaretçi sınıfının bir bölümü Tablo 25'te verilmiştir.

Tablo 25. Limit değerlendirme sınıfı

```
public class LimitVisitor implements ILimitVisitor {
    double a;
    public LimitVisitor(double n){ a = n; }
    …
    public double visit(Plus e)
    { return e.e1.accept(this)+ e.e2.accept(this); }
    …
    public double visit(X e) { return a; }
    public double visit(Num e) { return e.n; }
}
```

Tablo 25'deki değerlendirici sınıfının yapıcısında *a* değişkenine limit ifadesindeki *x*'in yakınsadığı *n* değeri atanır. *X* türünden parametre alan visit metodunda dönüş değeri olarak *a* üretilmesiyle limit fonksiyonunda bütün *x* değerlerine *n* değeri verilmiş olur.

Belirsizlik durumu oluşturmayan ifadeler için bu şekilde bir çözüm uygulanabilirken, belirsiz limit ifadeleri için öncelikle ifadedeki belirsizliğin tepit edilmesi gereklidir.

2.2.5. Belirsizlik Tespiti

Belirsizlik durumlarının tespiti ve uygulamada hata oluşturmalarını önlemek için bazı kontrollerin yapılması gerekmektedir. Ziyaretçi sınıfındaki ilgili visit metotlarına Tablo 26'deki şartların oluşması durumunda işlemin sonlandırılmasını sağlayan koşullar eklenerek sistem belirsiz limit durumlarının oluşturacağı hatalardan arındırılmalıdır.

Tablo 26. Belirsizlik yaratan işlemler

İŞLEM TÜRÜ	**e1**	**e2**
BÖLME	0	0
	∞	∞
ÇARPMA	0	∞
ÇIKARMA	∞	∞
ÜST ALMA	0	0
	1	∞
	∞	0

Tablo 27'de de görüldüğü gibi belirsiz durumlar AST ağacında sadece ilk 4 tip nesne ile başlayabilir. AST ağacı eğer bu nesnelerden biriyle başlıyorsa ağacın sağı ve solu ayrı ayrı değerlendirilerek (e1 ve e2) belirsizlik oluşturacak sonuçların çıkıp çıkmadığı kontrol

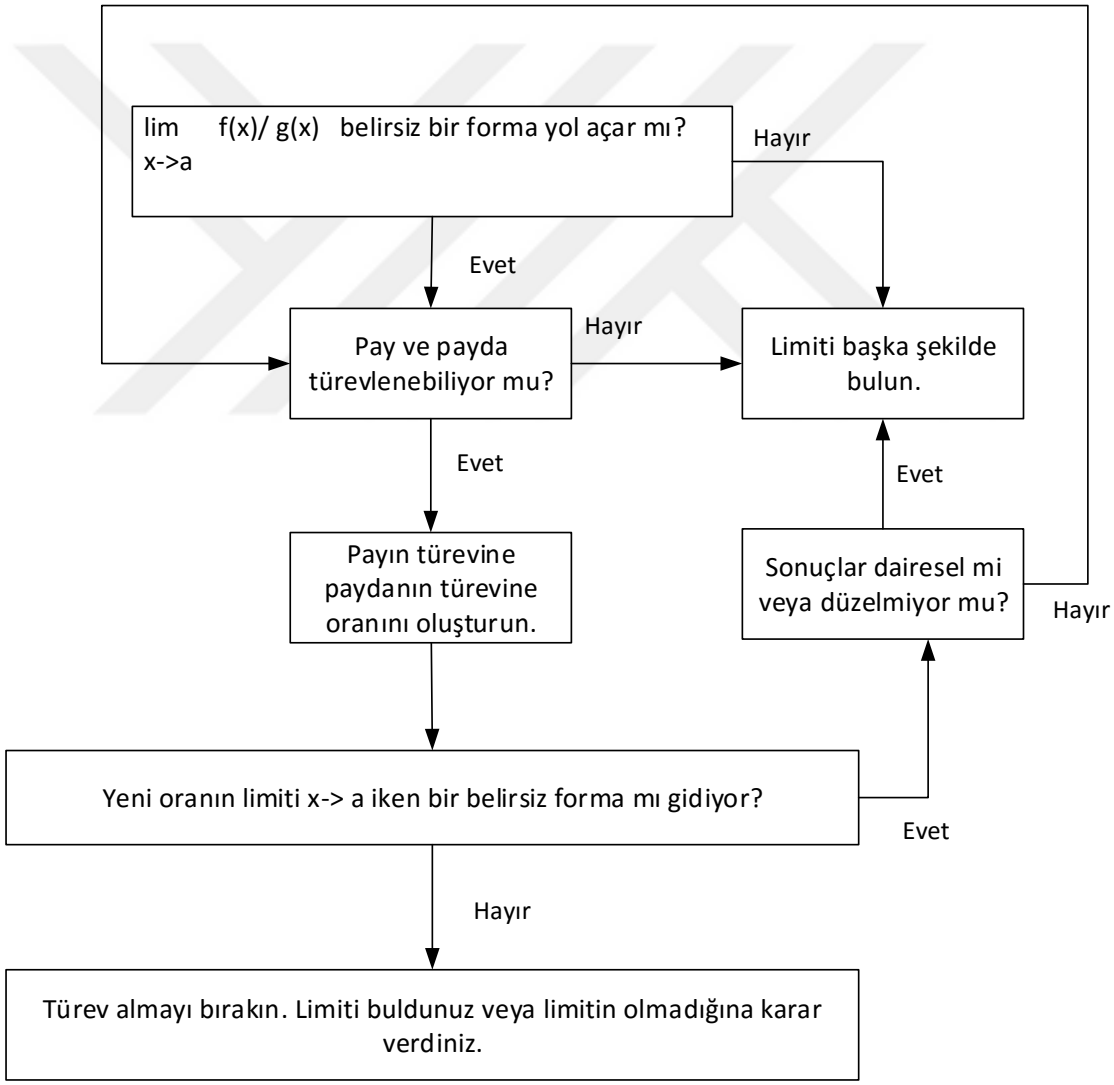
edilir. Değerlendirme sürecinin başında bu kontrolü yaparak hem belirsiz durumların hesaplama yapılırken oluşturacağı hatalar engellenir hem de belirsizlik tipleri belirlenmiş olur. Böylelikle sonraki adımda belirsizlik tiplerine göre çözüme devam edilir.

Tablo 27. Belirsizlik tiplerinin tespiti

```
public Value visit(Exp e) {
  Value a = ((Operator) e).e1.accept2(this);
  Value b = ((Operator) e).e2.accept2(this);
  if (e instanceof Divide) {
    if (a.type == Valuetype.Number && a.num == 0 && b.num == 0 &&
b.type == Valuetype.Number) {
         ... ←   0/0 formu
    } else if (a.type == Valuetype.Infinite && b.type ==
Valuetype.Infinite) {
         ... ←   inf/inf formu
    }
  }
  else if (e instanceof Power) {
    if (a.num == 1 && b.type == Valuetype.Infinite) {
         ... ←   1^inf formu
    } else if (a.type == Valuetype.Infinite && b.num == 0 &&
b.type == Valuetype.Number) {
         ... ←   inf^0 formu
    } else if (a.type == Valuetype.Number && b.num == 0 && b.type
== Valuetype.Number && b.num == 0 ) {
         ... ←   0^0 formu
    }
  }
  else if (e instanceof Minus) {
    if (a.type == Valuetype.Infinite && b.type ==
Valuetype.Infinite) {
         ... ←   inf-inf formu
    }
  }
  else if(e instanceof Times){
    if (a.type == Valuetype.Infinite && b.type ==
Valuetype.Number && b.num == 0) {
         ... ←   inf*0 formu
    } else if (b.type == Valuetype.Infinite && a.type ==
Valuetype.Number && a.num == 0) {
         ... ←   0*inf formu
    }
  }
  ...
}
```

2.2.6. Belirsizlik Çözümü

Belirsiz limit problemlerinin çözümünde birden fazla yöntem vardır. Özellikle diğer yöntemlerin simgesel hesaplama ile uygulanabilirliğinin polinom sadeleştirmenin zorluğuna takılmasından dolayı L'Hopital kuralı yöntem olarak tercih edilmiştir. Limit belirsizliği $0/0$ veya ∞/∞ biçiminde ise L'Hopital kuralına göre pay ve paydanın ayrı ayrı türevleri alınarak belirsizlik giderilir. Şekil 11'te çalışmada gerçekleştirilen belirsizlik çözümünün bir şeması yer almaktadır.



Şekil 11. Belirsizlik çözüm şeması

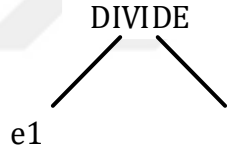
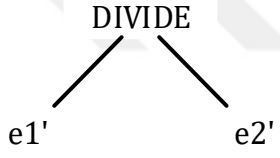
Limit ifadesine L'Hopital kuralı uygulayabilmek için öncelikle türev değerlendirme işlemini gerçekleştirecek değerlendiricinin geliştirilmesi gerekmektedir. Bu amaçla ziyaretçi tasarım şablonunu kullanarak türev için bir değerlendirme sınıfı oluşturulmuştur.

2.2.7. L'Hopital Kuralının Uygulaması

L'Hopital uygulanabilir biçimdeki ifadenin AST ağacı bölme işlemi ile başlar. Eğer bölme düğümünün sağ ve sol alt ağaçlarının değerlendirilmesi sonucunda ikisinden birden 0 veya ∞ sonucu geliyorsa L'Hopital uygulanabilir. Girdi olarak aldığı ağacının sağ ve sol dallarının türevini alarak çıkış olarak tekrar bir AST ağacı verir.

$f(x) = \frac{e1}{e2} \rightarrow e1 = e2 = 0 \;||\; e1 = e2 = \infty$ ise Tablo 28'deki gibi AST üstünde L'Hopital uygulanabilir.

Tablo 28. AST üstünde L'Hopital uygulanması

L'Hopital Giriş	L'Hopital Sonucu
	

2.2.8. Türev Alma

Türev işlemi AST ağacı değerlendirilerek yapılır. Bunun için ziyaretçi tasarım şablonuyla bir değerlendirme sınıfı oluşturulur. Bu sınıf girdi olarak aldığı ifadenin AST ağacınından yeni bir AST ağacı üretir.

Türev alma sınıfı oluşturulurken genel türev kurallarından faydalanılmıştır. Türev alma işlemini gerçekleştiren ziyaretçi sınıfındaki metotların bir kısmı gövdeleriyle birlikte Tablo 29'te verilmiştir.

Tablo 29. Türev alma sınıfı

Sınıf	İlgili Visit() metodu

Exp	`public Exp visit(Exp e) {`
	`return e.accept(this);`
	`}`

Plus	`public Exp visit(Plus e) {`
	`return new Plus(e.e1.accept(this), e.e2.accept(this));`
	`}`

Times	`public Exp visit(Times e) {`
	`return new Plus(new Times(e.e1.accept(this), e.e2), new Times(e.e1, e.e2.accept(this)));`
	`}`

Divide	`public Exp visit(Divide e) {`
	`Exp de1 = e.e1.accept(this);`
	`Exp de2 = e.e2.accept(this);`
	`Exp A = new Minus(new Times(de1, e.e2), new Times(e.e1, de2));`
	`Exp B = new Power(e.e2, new Num(2));`
	`return new Divide(A, B);`
	`}`

Power	`public Exp visit(Power e) {`
	`Exp de1 = e.e1.accept(this);`
	`Exp de2 = e.e2.accept(this);`
	`Exp A = new Times(new Times(e, de2), new Ln(e.e1));`
	`Exp B = new Power(e.e1, new Minus(e.e2, new Num(1)));`
	`return new Plus(A, new Times(new Times(e.e2, de1), B));`
	`}`

X	`public Exp visit(X e) {`
	`return new Num(1);`
	`}`

Sin	`public Exp visit(Sin e) {`
	`return new Times(e.e.accept(this), new Cos(e.e));`
	`}`

Üst operatörünün türevinin değerlendirilmesinde üstel ifadelerin türevlerini de kapsaması için genelleştirilmiş üst kuralından faydalanılmaktadır. Genel üst kuralı şu şekildedir:

$$(f^g)' = \left(e^{g \ln f}\right)' = f^g \left(f' \frac{g}{f} + g' \ln f\right) \tag{2.2}$$

Türev değerlendirici sınıfının örnek bir x^2 ifadesi üzerinden işlem adımları Tablo 30'da gösterilmiştir.

Tablo 30. x^2 ve x^xifadesi için türev alma işlem adımları

x^2	
Girdi	`x^2`
AST	`new Pow(new X(), new Num(2))`
Türev sonrası AST	`new Plus(new Times(new Times(new Pow(new X(),new Num(2)),new Num(0)), new Ln(new X())), new Times(new Times(new Num(2),new Num(1)), new Power(new X(), new Minus(new Num(2),new Num(1)))))`
x^x	
Girdi	`x^x`
AST	`new Pow(new X(), new X())`
Türev sonrası AST	`new Plus(new Times(new Times(new Pow(new X(),new X()),new Num(1)), new Ln(new X())), new Times(new Times(new X(),new Num(1)), new Power(new X(), new Minus(new X(),new Num(1)))))`

Türev visitor sınıfının çıkışındaki AST sadeleştirilmesi gereken birçok ifade içerebilir. Türev sonucu oluşması beklenen nesne "new Times(new Num(2), new X())" iken çok daha karmaşık bir nesne ağacı oluşmuştur. Bu karmaşıklığı gidermek için iki yol vardır:

- Türev alıcı meotları sadeleşebilir gereksiz ifadeler üretmeyecek şekilde yeniden yazmak,
- Verilen bir ifadeyi en sade haline getiren bir yorumlayıcı oluşturarak AST'yi türev sonrası yorumlatmak.

2.2.9. Sadeleştirme

Değerlendirme sürecinin ara adımlarınlarında veya yazdırma işlemi öncesinde ifade üzerinde sadeleştirme yapmak gerekebilir.

Her durum için ortak bir sadelik tanımı yapılamayacağından sadeleştirmenin tanımlanması oldukça karmaşıktır [22]. Bundan dolayı bütün matematiksel ifadeleri kapsayacak genel bir sadeleştirme süreci tanımlamak yerine sadeleştirme ifade veya problem türüne göre tanımlanmalıdır.

Sadeleştirme normalde kuralların yeniden yazılması ile yapılır. Bu amaçla dikkate alınması gereken çokça durum ve yeniden yazılması gereken çokça kural vardır. En basit haliyle ifadenin boyutunu azaltacak şekilde sadeleştirme kuralları düzenlenmelidir.

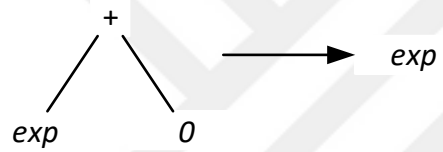
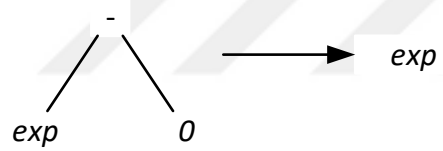
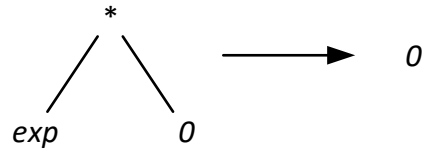
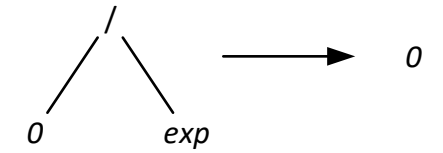
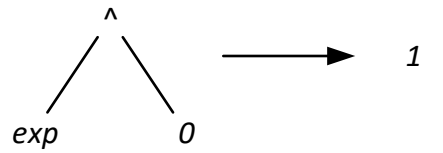
Örnek:

$E - E \rightarrow 0$

$sin(0) \rightarrow 0$

0 değerinin için işlemlere göre tanımlanan örnek sadeleştirme kuralları Tablo 31'da verilmiştir.

Tablo 31. Basit sadeleştirme kuralları için bazı dönüşümler

	new Plus(new X(), new Num(0))
	new X()
	new Minus(new X(), new Num(0))
	new X()
	new Times(new Num(0), new X())
	new Num(0)
	new Divide(new Num(0), new X())
	new Num(0)
	new Power(new X() ,0)
	new Num(1)

Tanımlanan sadeleştirme kurallarının örnek ifadeler üzerinde uygulanması sonucu Tablo 32'daki gibidir.

Tablo 32. Sadeleştirme örnekleri

İfade	Sadeleştirme sonucu
$x + 0$	x
$x + (-1)$	$x - 1$
$x + 3 - 1$	$x + 2$
$\frac{x}{1}$	x
x^0	1
x^1	x
$0 * x$	0
$1 * x$	x

Türev yorumlayıcısının x^2 ifadesi için oluşturduğu AST ağacının sadeleştirilip gereksiz ifadelerden kurtarılması gerekmektedir. Bu amaçla sadeleştirme kurallarına göre dönüşümler yapan sınıfımızın ifade üzerinde yaptığı değişiklikler Tablo 33'deki gibidir:

Tablo 33. Örnek sadeleştirme adımları

Giriş	`new Power(new X(), new Num(2))`
Türev	`new Plus(new Times(new Times(new Pow(new X(),new Num(2)),new Num(0)), new Ln(x)), new Times(new Times(new Num(2),new Num(1)), new Power(new X(), new Minus(new Num(2),new Num(1)))))`
1	`new Plus(new Times(new Times(new Pow(new X(),new Num(2)),new Num(0)), new Ln(x)), new Times(new Num(2), new Power(new X(), new Num(1))))`
2	`new Plus(new Times(new Times(new Pow(new X(),new Num(2)),new Num(0)), new Ln(x)), new Times(new Num(2), new X()))`
3	`new Plus(0, new Times(new Num(2), new X()))`
Çıkış	`new Times(new Num(2), new X())`

Birleşme özelliği:

Üç veya daha fazla ifadenin çarpımında, çarpılan ifadelerin herhangi iki tanesinin önce işleme alması sonucu değiştirmez. Bu duruma çarpma işleminin birleşme özelliği denir. Benzer şekilde toplama işlemininde birleşme özelliği vardır.

$$a * b * c = a * (b * c) = (a * b) * c$$
$$a + b + c = a + (b + c) = (a + b) + c$$

Değişme özelliği:

İki veya daha fazla ifadenin çarpımında çarpılan terimlerin yerlerinin değiştirilmesi işlemin sonucunu değiştirmez. Benzer şekilde toplama işleminde de olan bu özelliğe değişme özelliği denir.

$a * b = b * c$

$a + b = b + c$

Dağılma özelliği:

Bir terimi toplam biçimindeki bir ifade ile çarparken, bu terim ile toplam ifadesindeki her terim ayrı ayrı çarpılır ve ardından bu çarpımlar toplanır. Buna çarpma işleminin toplama işlemi üzerine dağılma özelliği denir. Benzer şekilde çarpma işleminin çıkarma işlemi üzerine de dağılma özelliği vardır.

$a * (b + c) = a * b + a * c$

$a * (b - c) = a * b - a * c$

Çarpanlara ayırma:

Çarpanlara ayırma ifadedeki tüm şartlar arasında yaygın olan faktörleri ortadan kaldırarak ifadeleri basitleştirmek için bir yoldur.

$a^2 - b^2 = (a + b)(a - b)$

Benzer Terimler:

Aynı dereceli aynı değişkenler içeren terimlere benzer terimler denir. Benzer terimlerin bir araya toplanması bu terimlerin birleştirilerek ifadenin sadeleştirilmesi için kolaylık sağlar.

Örnek:

- Benzer terimler: x , $2x, 5x$ vb.
- Benzer olmayan terimler: x, x^3 vb.

Benzer Terimlerin Birleştirilmesi:

Benzer terimler dağılma özelliği sebebiyle birleştirilebilir bu sayede daha sade bir ifade oluşturulur.

$$3x\text{^}2 + 5x\text{^}2 = (3 + 5)x\text{^}2 = 8x\text{^}2$$

Parantezler:

Benzer terimler birleştirilmeden önce varsa parantezli ifadelerin açılması gerekir. Burada da dağılma özelliğinden faydalanılır.

$$3x + 2(x - 4) = 3x + 2x - 8 = 5x - 8$$

Negatif İşaret:

Çıkarma işlemi ifadenin tersinin toplanmasıyla yer değiştirilebilir. Aynı şekilde negatif bir ifade toplama eklemek yerine tersi ile toplanması yer değiştirebilir.

$$3x - 2 = 3x + (-2)$$
$$3x + (-2) = 3x - 2$$

Negatif Kuvvet:

Bir ifadenin negatif kuvveti bölme ifadesi şeklinde de temsil edilebilir. Bu seheple payda da bulunan negatif kuvvetli ifade paya, paydaki ise paydaya taşınabilir.

$$\frac{2}{x^{-1}} = 2x$$
$$\frac{x^{-3}}{3^{-2}} = \frac{3^2}{x^3}$$

Kuvvetlerin Toplanması:

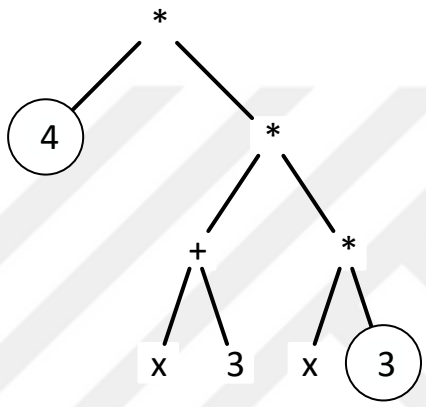
Üst alma işleminde tabanları aynı olan ifadelerin kuvvetleri toplanarak ifade kısaltılıp daha anlamlı hale getirilebilir.

$$x^2 * x^3 = x^5$$
$$(x + 1) * (x + 1)^3 = (x + 1)^4$$
$$\frac{3 * x^{-1}}{x^2} = 3/x^3$$

2.2.10. Sadeleştirme Sorunları:

AST üzerinde sadeleştirme işleminin yapılması bazı durumlarda daha zordur. Bunun bir sebebi her sadeleştirme işlemi ağacın yeniden düzenlenmesini gerektirir. Sadeleşecek ifadeler ağacın farklı dallarında farklı seviyelerinde olabilir.

Örnek $\mathbf{4} * (x + 1) * x * \mathbf{3}$ ifadesi için ayrıştırma sonrası oluşacak ağaç Şekil 12'da gösterilmiştir.



Şekil 12. Farklı seviyedeki benzer terimler

Örnekte de görüldüğü üzere 4 ve 3 ifadesi ağacın farklı seviyelerinde olabilmektedir. Bu tür ard arda gelen değişim özelliği bulunan operatörleri listelemek için Tablo 34'daki gibi bir *findTimes* methodu yazılabilir.

Tablo 34. Çarpma operatörlerini bulma metodu

```
public ArrayList<Exp> findTimes(Exp e){
    ArrayList<Exp> e1 = new ArrayList<Exp>();
    if(e instanceof Times){
        e1.addAll(findTimes(((Times) e).e1));
        e1.addAll(findTimes(((Times) e).e2));
    }else
        e1.add(e);
    return e1;
}
```

Örnek ifade için *findTimes* metodu Tablo 35'deki gibi kullanılabilir. Sonucunda *eList* değişkeni {4, Plus(x, 1), x, 3} listesini barındırır. Sadeleştirme uygulandığında liste {12, Plus(x, 1), x} şekline dönüşür. Sadeleştirilen ifade listesinden AST tekrar oluşturulur.

Tablo 35. FindTimes metodunun örnek kullanımı

```
ArrayList<Exp> eList = findTimes(e);
double carpim = 1;
for (int i = 0; i < eList.size(); i++)
    if (eList.get(i) instanceof Num) {
        carpim *= ((Num) eList.get(i)).n;
        eList.remove(i);
    }
if(carpim!=1) { eList.add(new Num(((int) carpim))); }
```

Sadeleşmeler bazı durumlarda sadece bir nesne değil alt nesne ağaçları olabilir. Bu tür sadeleştirmeleri gerçekleştirebilmek için iki alt ifadenin eşitliğinin belirlenmesi gerekir.

Örnek:

$$\frac{3(x+1)}{(x+1)x} = \frac{3}{x}$$

Tablo 36'de iki nesne ağacının eşitliğini özyinelemeli olarak kontrol eden örnek bir metod verilmiştir.

Tablo 36. İki ağacın eşitliğini değerlendiren metod

```
public static boolean isIdentical(Exp a, Exp b) {
    if (a == null && b == null) { return true; }
    if(a.getClass() != b.getClass()) { return false;}
    if (a instanceof Operator && b instanceof Operator) {
        return isIdentical (((Operator) a).e1, ((Operator) b).e1)
&& isIdentical (((Operator) a).e2, ((Operator) b).e2);
    } else if (a instanceof Function && b instanceof Function){
        return ((Function) a).e == ((Function) b).e;
    } else if (a instanceof Num && b instanceof Num){
        return ((Num) a).n == ((Num) b).n;
    } else if (a instanceof Variable && b instanceof Variable){
        return a.getClass().equals( b.getClass());
    }
    return false;
}
```

2.2.11. Trigonometrik Fonksiyonlar

Trigonometrik ifadeler simgesel hesaplama için değerlendirme, dönüştürme ve sadeleştirme işlemlerinde zorluk oluşturur. Bu tür ifadeler için dil bilgisi tanımlamaları yapılmalı, trigonometrik kurallar ve dönüşümleri yapacak algoritmaların geliştirilmesi gereklidir.

Trigonometrik ifadelerin kullanımı için programlama dillerinde sınıflar, kütüphaneler ve fonksiyonlar vardır. Bunlar genelde matematik fonksiyonlarıyla birlikte bulunurlar. Java programlama dilinde de Math sınıfı altında trigonometik ifadelerin değerini bulmak için methodlar vardır. Bu methodlar sayısal hesaplama yöntemleri ile geliştirilmiştir bu nedenle simgesel hesaplamada bu hazır metodlar kullanılamaz.

Tablo 37. Trigonometrik fonksiyonlar için sayısal ve simgesel hesaplama sonuçları

	Java Math Sınıfı Sayısal Sonucu	Simgesel Sonuç
Sin(30)	0.5000	$1/2$
Sin(45)	0.7071	$\sqrt{2}/2$
Sin(60)	0.8660	$\sqrt{3}/2$
Tan(90)	16331239353195370.0000	∞
Tan(30)	0.5774	$\sqrt{3}/3$

Trigonometrik fonksiyonların özel değerlerinin hesabı için sadece sinüs fonksiyonu için değerler tutulup diğer fonksiyonlar için sinüs üzerinden hesaplama yapılabileceği gibi Pisagor Teoreminden faydalanılarak birim üçgen üzerinden bu fonksiyon değerleri hesaplanabilir. Bunların dışında fonksiyonların tamamı için Tablo 38'deki gibi bir değer tablosundan faydalınabilir.

Tablo 38. Trigonometik değerler tablosu

Derece	Radyan	sin	cos	tan	cotan	sec	cosec
0°	0	0	1	0	∞	1	∞
30°	$\pi/6$	$1/2$	$\sqrt{3}/2$	$\sqrt{3}/3$	$\sqrt{3}$	$2\sqrt{3}/3$	2
45°	$\pi/4$	$\sqrt{2}/2$	$\sqrt{2}/2$	1	1	$\sqrt{2}$	$\sqrt{2}$
60°	$\pi/3$	$\sqrt{3}/2$	$1/2$	$\sqrt{3}$	$\sqrt{3}/3$	2	$2\sqrt{3}/3$
90°	$\pi/2$	1	0	∞	0	∞	1
120°	$2\pi/3$	$\sqrt{3}/2$	$-1/2$	$-\sqrt{3}$	$-\sqrt{3}/3$	-2	$2\sqrt{3}/3$
135°	$3\pi/4$	$\sqrt{2}/2$	$-\sqrt{2}/2$	-1	-1	$-\sqrt{2}$	$\sqrt{2}$
150°	$5\pi/6$	$1/2$	$-\sqrt{3}/2$	$-\sqrt{3}/3$	$-\sqrt{3}$	$-2\sqrt{3}/3$	2
180°	π	0	-1	0	∞	-1	∞
210°	$7\pi/6$	$-1/2$	$-\sqrt{3}/2$	$\sqrt{3}/3$	$\sqrt{3}$	-2	-2
225°	$5\pi/4$	$-\sqrt{2}/2$	$-\sqrt{2}/2$	1	1	$-\sqrt{2}$	$-\sqrt{2}$
240°	$4\pi/3$	$-\sqrt{3}/2$	$-1/2$	$\sqrt{3}$	$\sqrt{3}/3$	-2	$-2\sqrt{3}/3$
270°	$3\pi/2$	-1	0	∞	0	∞	-1
300°	$5\pi/3$	$-\sqrt{3}/2$	$1/2$	$-\sqrt{3}$	$-\sqrt{3}/3$	2	$-2\sqrt{3}/3$
315°	$7\pi/4$	$-\sqrt{2}/2$	$\sqrt{2}/2$	-1	-1	$\sqrt{2}$	$-\sqrt{2}$
330°	$11\pi/6$	$-1/2$	$\sqrt{3}/2$	$-\sqrt{3}/3$	$-\sqrt{3}$	$2\sqrt{3}/3$	-2
360°	2π	0	1	0	∞	1	∞

Trigonometrik fonksyionlar bir biri türünden yazılabilir. Bu durum bir birine eş sayılabilecek iki ifadenin ayrıştırılması sonucu iki farklı nesne ağacı oluşması sorununu doğurur.

Tümler Açılar:

Ölçüleri toplamı 90° olan (tümler) iki açıdan birinin sinüsü, diğerinin kosinüsüne; birinin tanjantı, diğerinin kotanjantına; birinin sekantı, diğerinin kosekantına eşittir.

$$\cos Q = \sin(90 - Q)$$
$$\sin Q = \cos(90 - Q)$$

Örnek:

$$\lim_{x \to a} \frac{\sin x}{\cos(90 - x)} = 1$$

Pisagor Teoremi:

Trigonometride Pisagor teoreminden yola çıkarak aşağıdaki eşitlikler yazılır. Bu eşitlikleri kullanarak x değişkeninin değerinden bağımsız olarak ifade içerisindeki trigonometrik fonksiyonlardan kurtulunur.

$$\cos^2 x + \sin^2 x = 1$$
$$\sec^2 x - \tan^2 x = 1$$
$$\csc^2 x - \cot^2 x = 1$$

Toplam ve Fark Formülleri:

Trigonometrik değerleri bilinen iki açının toplamının veya farkının trigonometrik değerlerini hesaplamak için kullanılan formüllerdir.

$$\cos(A - B) = \cos A \cos B + \sin A \sin B$$
$$\cos(A + B) = \cos A \cos B - \sin A \sin B$$
$$\sin(A - B) = \sin A \cos B - \cos A \sin B$$
$$\sin(A + B) = \sin A \cos B + \cos A \sin B$$

Yarım Açı Formülleri:

Yarım açı formülleri bir açı ile onun iki katının trigonometrik oranları arası ilişkiyi gösterir. Bu formüller toplam ve fark formüllerinden yola çıkılarak bulunmuştur.

$$\sin 2x = 2 \sin x \cos x = \frac{2 \tan x}{1 + \tan^2 x}$$
$$\cos 2x = \cos^2 x - \sin^2 x = 2 \cos^2 x - 1 = 1 - 2 \sin^2 x = \frac{1 - \tan^2 x}{1 + \tan^2 x}$$
$$\tan 2x = \frac{2 \tan x}{1 - \tan^2 x}$$
$$\cot 2x = \frac{\cot^2 x - 1}{2 \cot x}$$

Trigonometrik fonksiyonların bir birine dönüşebilmesinden dolayı birden fazla türev sonuçları olabilmektedir. Çözümü kolaylaştırma için L'Hopital uygulanırken alınan türev

sonucu hangi ifade ile devam edilmesi gerektiği önemli sorun teşkil eder. Örneğin aşağıda $\tan x$'in türevi için üç farklı ifade verilmiş olup bu örnekler artırılabilir.

$$(\tan x)' = \sec^2 x = \frac{1}{\cos^2 x} = 1 + \tan^2 x$$

$$(\cot x)' = -\csc^2 x = \frac{-1}{\sin^2 x} = -(1 + \cot^2 x)$$

2.2.12. L'Hopital Kuralının Sorunları

L'Hopital kuralı sadece teoremde belirtilen şatlarda uygulanabilir. Fakat bazı durumlarda L'Hopital uygulanabilir görünse de problemin çözümü için uygun değildir veya tek başına yeterli olmaz.

Bazı durumlarda L'Hopital kuralının sonlu defa ard arda uygulanması ile sonuca ulaşılamaz. Bu örnekte ifadeye ikinci defa L'Hopital uygulandığında ifade ilk haline dönüştüğü görülmektedir. Kural ard arda uygulandığında ifade iki biçim arasında döngüsel olarak dönüşür.

Örnek 1:

$$\lim_{x\to\infty} \frac{e^x + e^{-x}}{e^x - e^{-x}} = \lim_{x\to\infty} \frac{e^x - e^{-x}}{e^x + e^{-x}} = \lim_{x\to\infty} \frac{e^x + e^{-x}}{e^x - e^{-x}} = \cdots$$

Bu problemin çözümü için iki farklı şekilde yapılabilir:

1) İfadede e^x yerine y yazılır ve ifade y nin ∞'a yakınsaması şeklinde değiştirilir ve sonrasında L'Hopital uygulanarak çözüme gidilir.

$$\lim_{x\to\infty} \frac{e^x + e^{-x}}{e^x - e^{-x}} = \lim_{y\to\infty} \frac{y + y^{-1}}{y - y^{-1}} = \lim_{y\to\infty} \frac{1 - y^{-2}}{1 + y^{-2}} = \frac{1}{1} = 1$$

2) İfadenin pay ve paydası e^x'e bölünerek belirsizlik giderilir.

$$\lim_{x\to\infty} \frac{e^x + e^{-x}}{e^x - e^{-x}} = \lim_{x\to\infty} \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{1 - 0}{1 + 0} = 1$$

Örnek 2:

$$\lim_{x\to 0^+} x\ln x = \lim_{x\to 0^+} \frac{\ln x}{1/x} = \lim_{x\to 0^+} \frac{1/x}{-1/x^2} = \lim_{x\to 0^+} \frac{1/x^2}{-2/x^3} = \lim_{x\to 0^+} \frac{2/x^3}{-6/x^4} = \cdots$$

Bu problemin çözümünde L'Hopital uygulandıktan sonra ifade üzerinde sadeleştirmeye yapılarak belirsizlik giderilmiş olur.

$$\lim_{x\to 0^+} x\ln x = \lim_{x\to 0^+} \frac{\ln x}{1/x} = \lim_{x\to 0^+} \frac{1/x}{-1/x^2} = \lim_{n\to 0^+} -x = 0$$

Bazı durumlarda ise art arda L'Hopital uygulanması dairesel bir döngü yaratmasada belirsizliği gidermeyip ifadenin daha karmaşık hale gelmesine neden olur.

Örnek:

$$\lim_{x\to 0^+} x\ln x = \lim_{x\to 0^+} \frac{\ln x}{1/x} = \lim_{x\to 0^+} \frac{1/x}{-1/x^2} = \lim_{x\to 0^+} \frac{1/x^2}{-2/x^3} = \lim_{x\to 0^+} \frac{2/x^3}{-6/x^4} = \cdots$$

Bu problemin çözümü için değişken dönüşümü yöntemi kullanılabilir. İfadede x yerine $y = \sqrt{x}$ ifadesi koyulur ve L'Hopital kuralı uygulanarak çözüme ulaşılır.

$$\lim_{x\to\infty} \frac{x^{\frac{1}{2}} + x^{-\frac{1}{2}}}{x^{\frac{1}{2}} - x^{-\frac{1}{2}}} = \lim_{y\to\infty} \frac{y + y^{-1}}{y - y^{-1}} = \lim_{y\to\infty} \frac{1 - y^{-2}}{1 + y^{-2}} = \frac{1}{1} = 1$$

Bunlardan başka bir limit ifadesi içinde birden fazla belirsiz durum olabilir. Bu belirsiz ifadeler ayrı ayrı çözülüp değerleri ifade de yerine koyularak çözüme ulaşılır.

$$\lim_{x\to 0} \frac{\frac{\ln x}{x^{-1}}}{x^x} = \frac{\frac{\infty}{\infty}}{0^0} = \frac{1}{0} = \infty$$

2.2.13. Belirsizlik Dönüştürme

$0/0$ ve ∞/∞ biçimi dışındaki belirsizliklerin L'Hopital ile çözümü için öncelikle 0/0 veya ∞/∞ formuna dönüştürülmeleri gerekmektedir. Bunun için limit belirsizliği bölümündeki dönüşüm tablosundan yararlanılarak mevcut AST üstünde değişiklik yapılmalıdır.

Örnek bir $0 * \infty$ belirsiz formu için $f(x) = 0\, ise\, 1/f(x) = \infty$ yaklaşımından faydalanılarak yapılan belirsizlik dönüşümü Tablo 39'de gösterilmiştir.

Tablo 39. Örnek bir $0 * \infty$ belirsiz biçimi için belirsizlik dönüştürme

Giriş İfade	
$\lim_{x\to 0} \sin x \ln x$	$0 * \infty$
`new Times(new Sin(new X()), new Ln(new X()))`	
Dönüştürülmüş İfade	
$\lim_{x\to 0} \frac{\ln x}{(\sin x)^{-1}}$	$\frac{\infty}{\infty}$
`new Divide(new Ln(new X()), new Power(new Sin(new X()),new Num(-1)))`	

$0 * \infty$ biçimi için L'Hopital uygulanabilecek biçime dönüşümü yapacak algoritma basitçe Tablo 40'deki gibidir:

Tablo 40. Belirsizlik dönüştürme algoritması

```
e   ←   AST ağacının kökü
e1  ←   sol çocuk
e2  ←   sağ çocuk
if(e instanceof Times){
    Value a = e1.accept2(this);
    Value b = e2.accept2(this);
    if (a == Infinite && b == 0)   ← inf*0
        return new Divide(e1, new Power(e2, new Num(-1)));
    if (b == Infinite && a == 0)   ← 0*inf
        return new Divide(e2, new Power(e1, new Num(-1)));
}
```

Eğer ağaç Times nesnesi ile başlıyor ve sağ-sol dalları ayrı ayrı değerlendirildiklerinde ∞ ve 0 sonuçlarını veriyorlarsa $0 * \infty$ belirsiz formu vardır denilir. Bu belirsizliği ∞/∞ formuna dönüştürmek için;

- Bu ağacın kök düğümüne Divide nesnesi yerleştirir.
- Ağacın sol dalına inf sonucunu veren alt ağaç yerleştirilir.
- Ağacın sağ dalının soluna altında 0 sonucunu veren alt ağaç ve -1 olan Power nesnesi yerleştirilir.

$\infty - \infty$, 0^0, 1^∞ ve ∞^0 biçimdeki belirsiz bir ifadelerin L'Hopital uygulanabilir biçimdeki ifadelere dönüşümleri Tablo 41'da gösterilmiştir.

Tablo 41. Örnek belirsiz biçimleri için belirsizlik dönüştürme

Giriş İfade	
$\lim_{x\to 0} \frac{1}{\sin x} - \frac{1}{x}$	$\infty - \infty$
`new Minus(new Divide(new Num(1),new Sin(new X())),new Divide(new Num(1), new X()))`	
Dönüştürülmüş İfade	
$\lim_{x\to 0} \frac{x - \sin x}{x \sin x}$	$\frac{0}{0}$
`new Divide(new Minus(new X(), new Sin(new X())), new Times(new X(), new Sin(new X())))`	
Giriş İfade	
$\lim_{x\to 0} x^x$	0^0
`new Power(new X(),new X())`	
Dönüştürülmüş İfade	
$\lim_{x\to 0} x \ln x$	$0 * \infty$
`new Times(new X(), new Ln(new X()))`	
Dönüştürülmüş İfade	
$\lim_{x\to 0} \frac{\ln x}{x^{-1}}$	$\frac{\infty}{\infty}$
`new Divide(new Ln(new X()), new Power(new X(),new Num(-1)))`	
Giriş İfade	
$\lim_{x\to 0} \cos 2x^{\frac{1}{x}}$	1^∞
`new Power(new X(),new X())`	
Dönüştürülmüş İfade	
$\lim_{x\to 0} \frac{\ln \cos 2x}{x}$	$\frac{0}{0}$
`new Times(new X(), new Ln(new X()))`	
Giriş İfade	
$\lim_{x\to \infty} x^{\frac{1}{x}}$	∞^0
`new Power(new X(),new X())`	
Dönüştürülmüş İfade	
$\lim_{x\to 0} \frac{\ln x}{x}$	$\frac{\infty}{\infty}$
`new Times(new X(), new Ln(new X()))`	

2.2.14. İfadeyi Yazdırma

İfadeyi yazdırma işlemi AST ağacı üzerinde gezinerek her düğüm için belli bir metin formatta dönüş değeri üretilir. İfadeyi yazdırma işlemi aslında AST ağacının string haline dönüştürülmesidir.

Bu işlem için oluşturulan sınıf işlem adımları arasında kullanılarak adım adım çözüm yazdırılmış olur. Önceliklerin belirtilmesi için bazı ifadeler parantez içine alınarak yazdırılır.

Girdi : new Plus(new Times(new Num(2), new X()),new Num(5))

Çıktı : 2*x+5

İfadenin çözümünün ara adımlarındaki AST ağacının yazdırmak için oluşturulan sınıf Tablo 42'da verilmiştir.

Tablo 42. İfade yazdırma sınıfı

```
public class PrintVisitor implements IPrintVisitor {
    public String visit(Exp e) {
        return e.accept(this);
    }
    public String visit(Plus e) {
        String a = e.e1 instanceof Num || e.e1 instanceof X ?
e.e1.accept(this) : "(" + e.e1.accept(this) + ")";
        String b = e.e2 instanceof Num || e.e2 instanceof X ?
e.e2.accept(this) : "(" + e.e2.accept(this) + ")";
        return a + "+" + b;
    }
...
    public String visit(X e) {
        return "x";
    }
    public String visit(Num e) {
        return e.n < 0 ? "(" + e.n + ")" : "" + e.n;
    }
    public String visit(Ln e) {
        return "Ln(" + e.e.accept(this) + ")";
    }
    public String visit(Sin e) {
        return "Sin(" + e.e.accept(this) + ")";
    }
...
}
```

2.3. Belirsiz Limit İfadesi Üretme

Çalışmanın bu bölümünde otomatik belirsiz limit ifadeleri üreten bir sistem sunulmuştur. Bir önceki aşamada limit ifadelerinin dilbilgisi çıkarımı yapılmıştır. Bu sebeple bu aşamada çeşitli kaynaklardan belirsiz limit problemler örnekleri incelenmiş, bu tür problemlerin en genel şekli ile nasıl üretilebileceği üzerinde çalışılmıştır.

Yapılan çalışma şu ana başlıklardan oluşmaktadır:

- Rastgele ifade üretimi yöntemleri
- Belirsiz limit ifadeleri üretimi
- Üretilen ifadenin sadeleştirilmesi ve düzenlenmesi
- İfadelerin yazdırılması

2.3.1. Rastgele İfade Üretme

Otomatik belirsiz limit ifadesi üretimi rastgele matematiksel ifade üretiminin bir alt problemi olarak ele alınabilir. Rastgele ifade üretmek için birçok farklı yöntem olmakla beraber en basit yaklaşım Tablo 43'de verilen şekilde olabilir:

Tablo 43. Basit rastgele cebirsel ifade üretme algoritması

```
exp ← rastgele sayi üretilir
loop
     oper← rastgele sayi üretilir
     n   ← rastgele sayi üretilir
     exp ← exp + operator + n
endloop
...
```

Bu algoritma, rastgele üretilen iki sayının arasına [+,-,*,/] kümesinden rastgele seçilen bir operatörün koyulması ile ifade üretimi yapar.

Rastgele ifadeler Java'nın *Random* sınıfından yararlanarak üretilir. Rastgelelik sağlamak için öncelikle ihtimaller belirlenir. Sonrasında 0 ile muhtemel ihtimal sayısı aralığında Random sınıfın nextInt metodu ile rastgele sayı üretilir. Bu üretilen sayıya karşılık düşen ihtimal seçilerek devam edilir. Tablo 44'de Java'da rastgele sayı üretme kod parçası yer almaktadır.

Tablo 44. Java rastgele sayı üretme

```
Random generator = new Random();
int n = generator.nextInt(6);
```

Java Random sınıfı kullanarak rastgele belirsiz limit ifadesi üretmenin genel algoritması Tablo 45'daki gibidir.

Tablo 45. Rastgele belirsiz limit üretme algoritması

```
n ← 0..6 rastgele sayi üretilir
switch n
     case 0:
     0/0 üret
     case 1:
     ∞/∞ üret
     case 2:
…
```

Belirsizlik tipleri 0 ile 6 arasındaki rakamları ile eşleştirilir. Hangi tür belirsizlik tipinden soru üretileceğinin rastgele belirlenmesi için 0-6 arasında bir tam sayı, rastgele sayı üreteci ile üretilir. Üretilen sayının değerine karşılık düşen belirsizlik formunda problem üretimi yapılır.

Belirsizlik tipi rastgele değilde kullanıcı tanımlı yapılmak istendiğinde n rakamı kullanıcının seçimine göre atanır ve diğer kısımlarda değişiklik yapılması gerekmez.

Rastgele soru üretmek için dil bilgisini kullanarak string ifade üretmek ile rastgele nesne ağacı üretmek olarak iki yaklaşımdan söz edilebilir:

2.3.2. Dil Bilgisi Kuralı Tabanlı İfade Üretme

Dil bilgisi düzeyinde rastgele ifade üretimi şeklidir. Öncelikle dilin analizinin yapılıp dil bilgisi kurallarının oluşturulması gerekir. Mevcut dil bilgisi kurallarına göre bir sonraki parçanın neler olabileceği belli olduğundan olası seçenekler arasından rastgele seçilir. Toplama ve çıkarma operatöründen oluşan iki işlemli bir ifadeler için örnek dil bilgisi aşağıdaki gibi yazılabilir.

E → T E’

E’ → (+|-) T E’

E’ →

T → Num

Tablo 46’de bu dil bilgisine dayalı otomatik rastgele ifade üretme adımları gösterilmiştir.

Tablo 46. Dilbilgisi tabanlı ifade üretme

Adım	**İşlem**
1	Dil bilgisi bir *Num* türü başlamak zorunda olduğundan rastgele bir sayı seçilir
	2
2	Devamında dil bilgisine göre + ya da – gelebileceğinden rastgele biri seçilir.
	2+
3	2+8
4	2+7-3
...	2+7-3…
Sonuç	Başlangıçta belirlenmiş uzunlukta ifade üretimi durdurulur

Dil bilgisi kuralı tabanlı üretim basit fonksiyon ifadeleri için uygun olabilir. Fakat üretimden sonra ifadenin değerlendirilip çözümlü olup olmadığı kontrol edilmelidir. Belirsiz limit ifadesi üretiminde de dil bilgisi tabanlı üretimler belirsiz ifade oluşacağını garanti edemez. Bu özelliklerinden dolayı belirsiz limit problemi üretmeye pek uygun değildir.

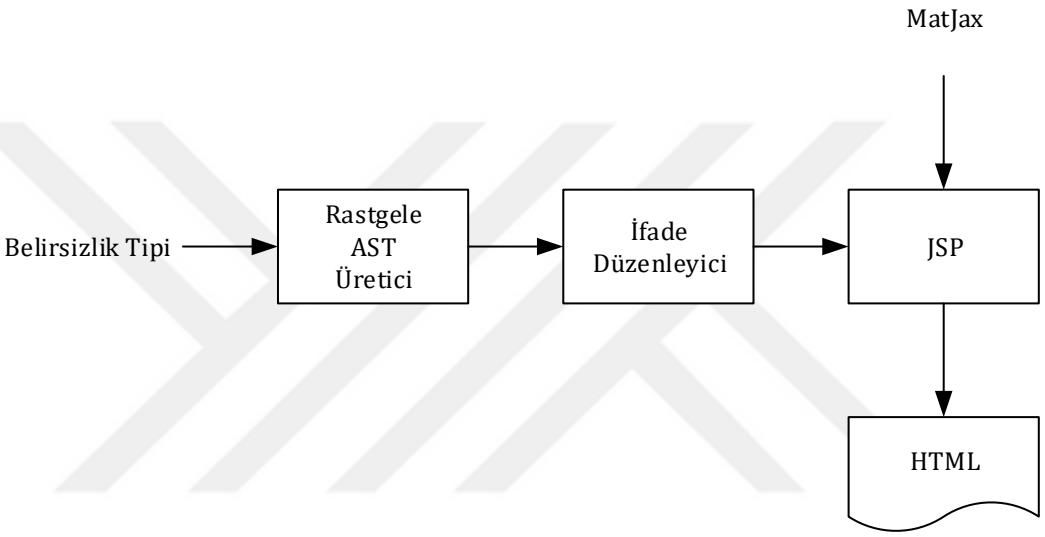
2.3.3. AST Tabanlı İfade Üretme

Bu yöntemde ifade üretimi rastgele AST ağacı oluşturarak gerçekleştirilir. Genel olarak şu aşamalardan oluşur.

- Kök düğümü için bir nesne rastgele seçilir ve yerleştirilir.
- Alt düğümler içinde rastgele nesneler belli bir düğüm sayısına veya AST ağacı belli derinliğe ulaşana kadar yerleştirilir.

- Oluşan AST ağacı ifadenin daha düzenli görünmesi için sadeleşme işlemi uygulanır.
- Sadeleşen AST ağacı düzenlenip yazdırılır.

Belirsiz limit ifadesi üretimi için tasarlanan sistemin genel yapısı Şekil 13'de gösterilmiştir.



Şekil 13. Rastgele AST üretme şeması

İlk aşama olan rastgele AST üretiminde oluşturulacak ifadenin karmaşıklığı, zorluğu, uzunluğu belli parametrelere bağlanabilir;

- Çıktı ifadenin uzunluğu
- AST'deki düğüm sayısı
- AST'deki seviye sayısı

Limit problemi üretme aşağıdaki adımlardan oluşur:

1. Problem üretme işlemi hangi tip belirsizlik türünden örnek üreteceği belirlenerek başlar. Bu aşamada 7 tip belirsizlikten rastgele biri seçilebilirken kullanıcı da seçim yapabilir. Belirsizlik tipi belirlendiğinde ağacın kök düğümünün hangi nesne olacağı da belirlenmiş olur.
2. Limit ifadesinde x değerinin yakınsayacağı a değeri rastgele seçilir.

3. Belirsizlik tipine göre kökün sağ ve sol alt ağaçlarında hangi sonuçların ne olması gerektiği bellidir. Buna göre 0, 1 veya ∞ üretecek alt ağaçlar oluşturulur.
4. Bir önceki aşamada üretilen ağaçlar belirsizlik tipine göre bir kök düğümü altına yerleştirilir.

Rastgele belirsiz limit ifadesi üretme adımları Tablo 47'de gösterilmiştir.

Tablo 47. Örnek bir belirsiz ifade üretme aşamaları

İşlem Adımı	**İşlem Sonucu**
Üretilecek belirsizlik türü	$P(x)/Q(x) = 0/0$
AST kök düğümü	`new Divide(?, ?)`
Rastgele a üret	`a=3`
Rastgele $P(x) = 0$ üret	`P = new Minus(new X(), new Num(3))` $x - 3$
Rastgele $Q(x) = 0$ üret	`Q = new Ln(new Minus(new X(), new Num(2)))` $ln(x - 2)$
Birleştirme	`new Divide(P,Q)` $\frac{x-3}{ln(x-2)}$
Sonuç	$\lim_{x \to 3} \frac{x-3}{ln(x-2)}$

Belirsiz formlar incelendiğinde 0, 1 ve ∞ sonuçlu ifadelerden oluştukları görülmektedir. Belirsizliği bu değerli ifadelerin çarpma, bölme, çıkarma ve üst alma operatörleri ile kullanıldığına oluştuğundan yola çıkarak, belirsiz bir ifadeyi tek seferde üretmek yerine önce 0, 1 ve ∞ sonuçlu ifadeler üretip bunları uygun bir operatör altına yerleştirerekte ifade üretilebilir. Bu şekilde problem çözümlenmesi daha basit üç alt parçaya ayrılmış olur. Buradan yola çıkarak rastgele 0, 1 ve ∞ değerli ifadelerin nasıl üretilebileceğinin incelenmesi gerekmektedir.

2.3.3.1. Sıfır Değerli İfade Üretmek

Sıfır sonucu veren ifadeler üretmek için hangi işlemler sonucunda 0 elde edildiğinin incelenmesi gerekir. Bu işlemler Öncelikle 0 değerini kullanmadan hangi işlem ve fonksiyonlardan 0 sonucu üretilebileceği Tablo 48'de verilmiştir.

Tablo 48. Sıfır sonucu veren işlemler

$n - n = 0$	`new Minus(new Num(n), new Num(n))`
$n + (-n) = 0$	`new Plus(new Num(n), new Num(-n))`
$\frac{n}{\infty} = 0$	`new Divide(new Num(n), inf)`
$\ln 1 = 0$	`new Ln(new Num(1))`
$\cos 90 = 0$	`new Cos(new Num(90))`
$\operatorname{cotan} 90 = 0$	`new CoTan(new Num(90))`

Bunun yanında 0 değerinin herhangi bir ifade ile bazı işlemlere tabi tutulması sonucunda yine 0 değeri veren ifadeler üretilebilir. Bu işlemler Tablo 49'de verilmiştir.

Tablo 49. Sıfır ile işleme girince 0 sonucu veren işlemler

$\frac{0}{n} = 0 \,; n \neq 0$	`new Divide(new Num(0), ?)`
$0^n = 0 \,; n \neq 0$	`new Power(new Num(0), ?)`
$0 * n = 0 \,; n \neq \infty$	`new Times(new Num(0), ?)`
$\sin 0 = 0$	`new Sin(new Num(0))`
$\tan 0 = 0$	`new Tan(new Num(0))`

Limit ifadeleri incelendiğinde genellikle 0 sonuçlarını veren ifadelerin x değişkenine bağlı oldukları görülmektedir. Bunu sağlayacak şekilde otomatik üretilecek ifadelerde X değişkenine bağlı olarak 0 değeri alacak nesne ağaçları Tablo 50'da verilmiştir.

Tablo 50. X ifadesine bağlı 0 üretmek

$\frac{0}{x}; x \neq 0$	`new Divide(new Num(0), new X())`
$0^x; x \neq 0$	`new Power(new Num(0), new X())`
$0 * x; x \neq \infty$	`new Times(new Num(0), new X())`
$x - n;\ x \neq \infty\ \&\ n = x$	`new Minus(new Num(n), new X())`

Yukarıdaki ifadelerden yola çıkarak 0 sonucu veren ifade üreten metot gövdesi aşağıda verilmiştir.

Tablo 51. Sıfır sonuçlu ifade üreten metot

```
public Exp zero(int lvl) {...
Exp e = null;
Exp zero = zerowithX();
int t = rand.nextInt(7);
switch (t) {
     case 0:
     e = new Divide(zero, expgen());
     break;
     case 1:
     e = new Power(zero, expgen());
     break;
     case 2:
     e = new Times(new Num(3),new Times(zero, expgen()));
     break;
     case 3:
     Exp e2 = expgen();
     e = new Minus(e2, e2);
     break;
     case 4:
     e = n!=0 ? new Sin(zero):new Sin(new
Times(zero,expwithN(1)));
     break;
     case 5:
     e = new Cos(new Num(90));
     break;
     case 6:
     e = new Ln(new Num(1));
     break;
}
return e;
}
```

2.3.3.2. Sonsuz Değerli İfade Üretmek

Sonsuz ifadesi bölme ve üst alma işlemlerinin sonucunda oluşur. Dolayısıyla değerlendirildiğinde sonsuz çıkan bir ifade üretmek istediğimizde ağacın başlangıç düğümü Power ya da Divide nesnesi olmalıdır. Bu ifadeler Tablo 52'da verilmiştir.

Tablo 52. Sonucu sonsuz olan işlemler

$1/0 = \infty$	$0^{-1} = \infty$
$n + \infty = \infty$	$n - \infty = -\infty$
$n * \infty = \infty$	$\frac{\infty}{n} = \infty$

Yukarıdaki tablodan faydalanılarak sonsuz sonuçlu ifade üreten bir java kod parçası aşağıdaki gibidir:

Tablo 53. Sonsuz sonuçlu ifade üreten metot

```
public Exp inf() {
    Exp e = null;
    int t = rand.nextInt(1);
    if (t == 0) {
        e = new Divide(expgen(), zero(0));
    }
    else {
        e = new Power(zero(0), new Times(new Num(-1),
expwithN(2)));
    }
    return e;
}
```

2.3.4. İfade Düzenleme

Rastgele problem üretiminde anlamsal olarak sorun teşkil etmeyen fakat anlaşılabilirlik ve gereksiz ifadelerden kurtarmak için ifadelerin düzenlenmesi gerekebilir. Bu tür ifadelerin düzenlenmesi için önceki bölümde oluşturulan sadeleştirici kullanılır. Burada dikkat edilmesi gereken husus bazı durumlarda yapılan sadeleştirmelerin ifadedeki belirsizliği ortadan kaldırabilmektedir.

Örnek:

$$\lim_{x\to 0} \frac{2x}{x^{-1}} = \frac{0}{0}$$

$$\lim_{x\to 0} 3x = 0$$

Ayrıca rastgele üretilen ifade anlamı değişmeyecek şekilde matematikteki genel yazım formatına çevirilmesi gerekir. Tablo 54'de örnek düzenlemelerden birkaçı verilmiştir.

Tablo 54. İfade düzenleme

İfadenin ilk hali	Düzenleme sonrası
$1 + x$	$x + 1$
$x * 2$	$2 * x$
$1 + x + x^2$	$x^2 + x + 1$

Matematiksel yazımda ifadeler büyük dereceli değişkenden daha küçük derecereye doğru sıralama ile yazılır. Ayrıca değişkenlerin kat sayıları solunda olacak şekilde yazılmalıdır. Bu kurallara göre AST üzerinde değişiklikler yapılır. Bu amaçla sadeleştirme sınıfının çarpma ile ilgili metodunda aşağıdaki gibi bir düzenleme eklenebilir.

Tablo 55. İfade düzenleme kodu

```
public Exp visit(Times e) {
...
Exp a = e.e1.accept(this);
Exp b = e.e2.accept(this);

if (b instanceof Num){
    return new Times(b, a);}
else
    return new Times(a, b);
...
}
```

Girdi: new Times(new X(), new Num(2))

Çıktı: new Times(new Num(2), new X())

2.3.5. İfade Yazdırma

Rastgele değerlerle üretilen belirsiz limit problemleri çalışmamızın ilk bölümündeki adım adım belirsiz limit problemleri çözümünde Tablo 42'da verilen *PrintVisitor* yazdırıcı sınıf ile metin formatta çıktısı oluşturulur.

2.4. Sistem Testleri

Yazılım geliştirmede dikkat edilmesi gereken konulardan biri de hatasız bir sistem geliştirmedir. Bu amaçla geliştirilen sistem çeşitli yazılım test süreçlerinden geçirilmelidir.

Yazılım testi, sınırlı sayıda ve uygun şekilde seçilmiş test kümesi için yazılımın ürettiği sonuçlar ile beklenen sonuçların karşılaştırılmasıdır. Bu süreci kolaylaştırmak ve standartlaştırmak için birçok yöntem, metot ve teknoloji geliştirilmiştir. Çalışmamızda Java dili için geliştirilen birim seviyede testler oluşturmaya imkân veren JUnit teknolojisi kullanılmıştır. Bu amaçla geliştirilen test sınıfının bir kısmı Tablo 56'de verilmiştir.

Tablo 56. Geliştirilen test sınıfı

```
public class MyTest {
@Test
public void SimplificationTest() {
     LimitVisitor limitVisitor = new LimitVisitor();
     Exp input,result;

     input = new Divide(new Num(2),new Num(4));
     result = new Divide(new Num(1), new Num(2));
     assertTrue(isIdentical(limitVisitor.visit(input), result));

     input = new Divide(new Times(new Num(4),new X()),  new X());
     result = new Num(4);
     assertTrue(isIdentical(limitVisitor.visit(input), result));

     input = new Times(new Times(new Num(2),new X()), new X());
     result = new Times(new Num(2),new Power(new X(),new Num(2)));
     assertTrue(isIdentical(limitVisitor.visit(input), result));
     ...
}
}
```

Geliştirilen türev, sadeleştirme, değerlendirme sınıflarının her biri için farklı örnekleri kapsayacak testler oluşturulması, yazılım geliştirme sürecinde oluşabilecek hataların tespit edilmesine imkân sağlamıştır.

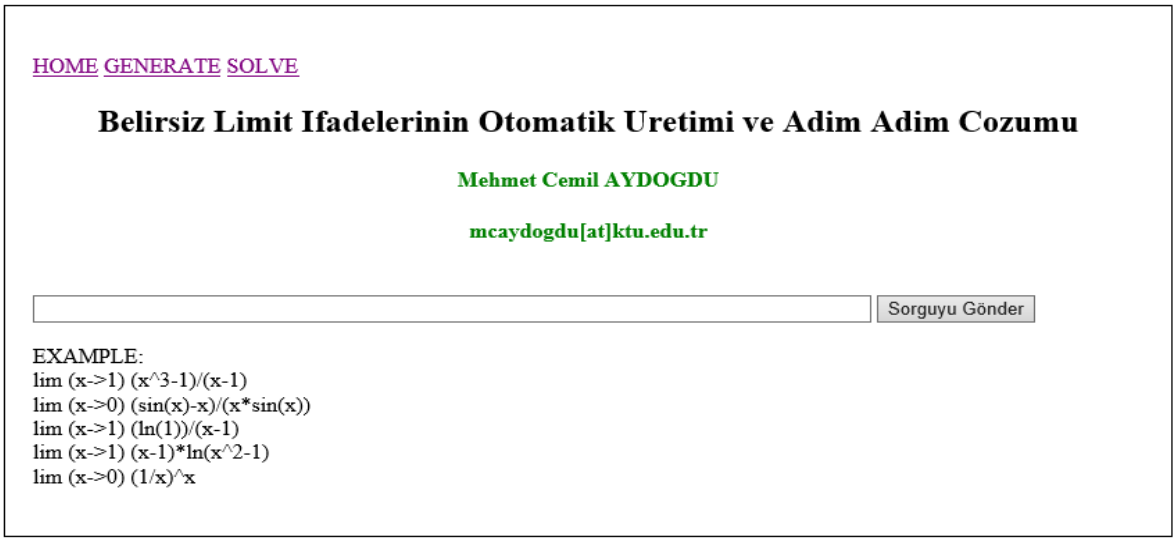
2.5. Kullanıcı Arayüzü

Çalışmamızda kullanıcı arayüzü olarak web platformu tercih edilmiştir. Çalışmamızda geliştirilen limit soruları için adım adım çözüm gerçekleştirmek ve rastgele belirsiz limit problemi oluşturan program bir jar formatında paketlenmiştir. Bu sayede herhangi bir projeye eklenip kullanılabilir durumdadır. Bu özellikten faydalanarak ayrı bir java web projesi oluşturulmuş ve bu parser-generator.jar paketi bu projeye eklenerek kullanılmıştır.

Web projesini geliştirmek için JSP teknolojisi tercih edilmiş ve tomcat uygulama sunucusunda sunulmuştur.

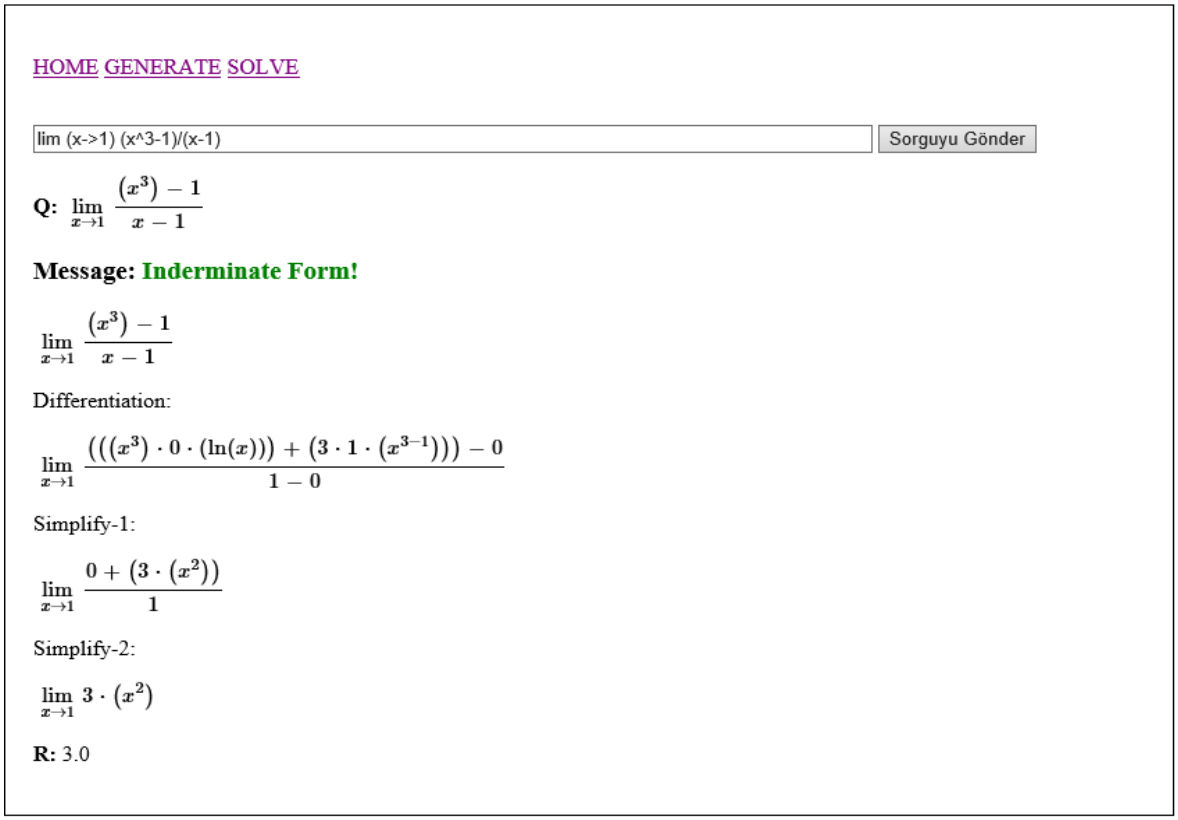
Uygulamamızda matematiksel ifadeler string formunda kullanılmıştır. Anlaşılırlığı ve sunumu güzelleştirmek açısından MathJAX javascript kütüphanesi ile bu çıktılar kullanıcı arayüzünde matematiksel gösterim şekliyle sunulmuştur.

Geliştirilen kullanıcı arayüzününde sistemin açılış sayfasının ekran çıktısı Şekil 14'de verilmiştir.



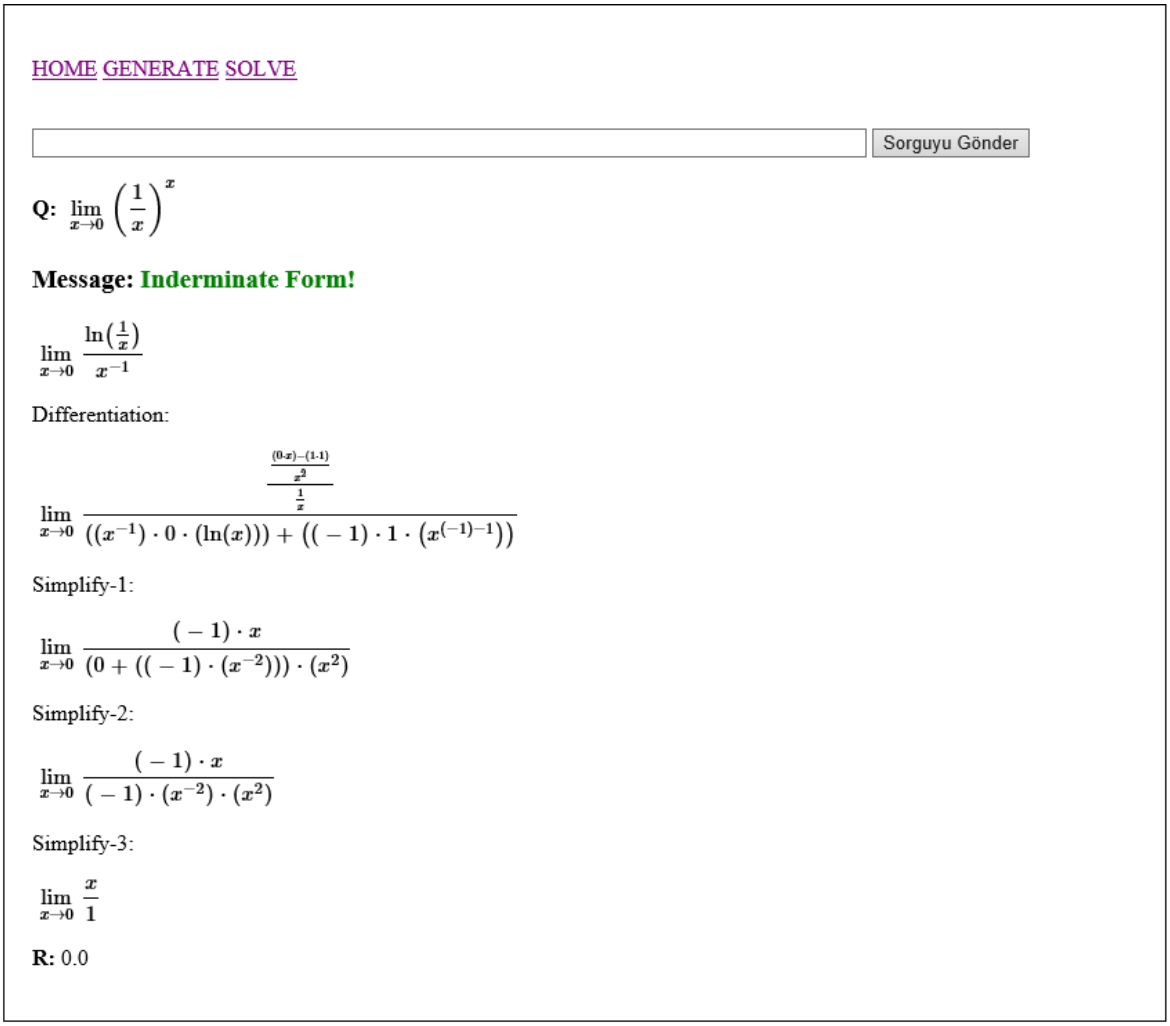
Şekil 14. Sistemin giriş ekranı

Sistemin “lim (x->1) (x^3-1)/(x-1)” belirsiz ifadesi için oluşturduğu çözümün ekran çıktısı Şekil 15’te verilmiştir.



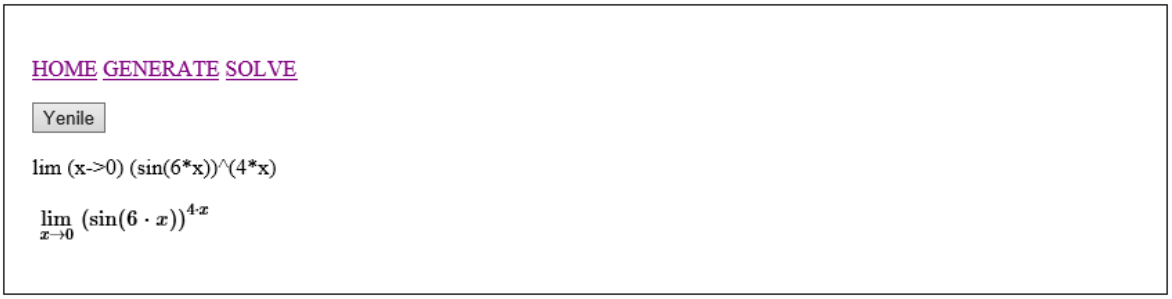
Şekil 15. Örnek $0/0$ biçimdeki bir belirsiz limit ifadesinin çözümü

Sistemin “lim (x->0) (1/x)^x” belirsiz ifadesi için oluşturduğu çözümün ekran çıktısı Şekil 16’te verilmiştir.



Şekil 16. Örnek ∞^0 biçimdeki bir belirsiz limit ifadesinin çözümü

Sistem üzerinde otomatik belirsiz limit problemi sayfasında her yenileme farklı bir ifade üretilecek şekilde bir arayüz tasarlanmıştır ve bu sayfanın ekran çıktısı Şekil 17'de verilmiştir.



Şekil 17. Otomatik belirsiz limit ifadesi üretimi

3. SONUÇLAR VE TARTIŞMA

Çalışmada limit problemlerinin tamamını kapsayacak bir dilbilgisi grameri tanımlanmıştır. Bununla birlikte basit yapıdaki limit sorularının çözümünü yapan, ayrıca varsa belirsiz durumları ve belirsizlik türünü tespit eden yorumlayıcı bir Java sınıfı oluşturulmuştur. Tablo 57 ve Tablo 58'de iki farklı örnek için değerlendirme sürecinin aşamaları kısaca gösterilmiştir.

Tablo 57. Örnek bir basit limit problemi çözüm adımları

Problem	$\lim_{x\to 2} \frac{x+2}{x-1}$
Girdi	`lim (x->2) (x+2)/(x-1)`
AST	`new Divide(new Plus(new X(),new Num(2)),new Minus(new X(),new Num(1))`
Değerlendir	`(2 + 2) / (2 - 1)`
Sonuç	`4 / 1 -> Belirsiz Form Değil` `Sonuç = 4`

Tablo 57'de basit bir limit örneği için işlem adımları verilmiştir. Limit ifadesi belirsiz bir form içermediği için sonuç hesaplanabilmektedir. Tablo 58'de belirsiz formdaki ifadenin değerlendirilmesi için işlem adımları gösterilmiştir.

Tablo 58. Örnek bir belirsiz limit problemi için çözüm adımları

Problem	$\lim_{x\to 1} \frac{\ln(x)}{x-1}$
Girdi	`lim (x->1) ln(x)/(x-1)`
AST	`new Divide(new Ln(new X()),new Minus(new X(),new Num(1))`
Değerlendir	`ln(1) / (1 - 1)`
Ara Sonuç	`0 / 0 -> [0/0] Belirsiz Form`
Türev Al	`new Divide(new Divide(new Num(1),new X()),new Num(1))`
Sadeleştir	`new Divide(new Num(1),new X())`
Değerlendir	`1/1`
Sonuç	`1`

Çalışmamız bütün basit limit problemleri için çözüm üretebilirken, belirsiz formdaki limit problemleri için önce belirsizlik tipi tespiti yapmaktadır. Bu aşamadan sonra belirsizlik uygun formda ise L'Hopital teoremi doğrudan uygulanabilirken, uygun formda olmayan belirsizlikler için öncelikle dönüştürme yapılmaktadır.

Çalışmamız sonucunda L'Hopital teoremi ile çözülebilen bütün belirsiz limit problemlerinin adım adım çözümü üretilebilmektedir. Fakat bazı problemlerde ard arda L'Hopital uygulanması ile belirsizliğin giderilemediği ve özel dönüşümler ve sadeleştirmeler gerektiği görülmüştür. Bu tür problemlere özel farklı çözüm yolları üzerinde araştırma yapılması gerekir.

İkinci bölümde ise otomatik belirsiz limit ifadeleri üretimi üzerine çalışılmıştır. Otomatik problem üretimi için literatürde dil bilgisi tabanlı yöntemler mevcuttur. Fakat belirsizlik durumu dil bilgisi seviyesinde değil, ifade değerlendirme sürecinde tespit edilebilen bir durumdur. Bu nedenle literatürdeki otomatik ifade üretim yöntemlerinin belirsiz limit ifadesi üretimi için pek uygun olmadığı görülmüş ve çalışmamızda bu problemlerin üretimi için belirsiz durum oluşturacak rastgele AST oluşturma yöntemi kullanılmıştır.

Belirsiz form yapıları incelendiğinde sonsuz, sıfır ve bir değerli alt ifadelerden oluştukları görülmüştür. Bu alt ifadeler rastgele üretilip, bir operatör altında bir araya gelmesi ile istenen belirsiz form üretilmiş olur. Limit soruları, konu hakkındaki soru bankaları ve çözümlü sorulardan incelendiğinde limit fonksiyonlarının karmaşıklığının düşük olduğu ve problemlerin amacının fonksiyonun çözülmesinden çok limit konusunun kurallarının, L'Hopital kuralının ve türev konusunun pekiştirilmesi olduğu görülmektedir. Bu amaca uyacak şekilde uygulama ile rastgele oluşturulacak fonksiyonun derecesi ve karmaşıklığı yüksek olmamalıdır.

Sunulan yöntem ile bütün belirsiz form tiplerinden otomatik ifade üretimi yapılabilirken, üretilen ifadenin karmaşıklığı belli parametrelere bağlanmıştır.

Genel olarak değerlendirdiğimizde geliştirilen sistemin özellikler aşağıdaki gibidir:

- Limit ifadelerini ayrıştırabilmektedir.
- Belirsiz limit ifadelerini L'Hopital uygulayarak adım adım çözümünü yapabilmektedir.
- İfadelerin türevini simgesel olarak hesaplayabilmektedir.
- İfadeler üzerinde basit sadeleştirmeler yapabilmektedir.
- Otomatik belirsiz limit ifadeleri üretebilmektedir.

- Matematiksel ifadelerin gösterimini yapabilmektedir.
- Birim test alt yapısı sayesinde birim seviyesinde testler yazılabilmektedir.
- Bir framework olarak başka projelere eklenebilmektedir.

Çalışmanın diğer mevcut örneklerle karşılaştırılması Tablo 59'deki gibidir.

Tablo 59. Diğer sistemlerle karşılaştırma

	Wolfram	**Matlab Sembolik**	**Çalışmamız**
Adım adım çözüm	Var	Yok	Var
Framework	Yok	Yok	Var
Otomatik İfade üretimi	Var(Ücretli versiyonda)	Yok	Var

Sonuç olarak simgesel programlama sistemleri, uzun ve karmaşık hesapların yapılması, kesin bir sonuç üretebilmesi nedeniyle araştırmalarda önemli kolaylıklar sağladığı gibi bilgisayar destekli eğitim alanına da katkıda bulunmuştur. Bu açıdan kullanım alanının giderek artacağı açıktır [47]. Bizim çalışmamızda otomatik kod üretim araçlarını kullanarak belirsiz limit ifadelerinin bilgisayar destekli adım adım çözümü ve otomatik üretimi konusunda bir simgesel hesaplama çalışması örneğidir. Bu nedenlerden ötürü çalışmamızın sonraki araştırmalara ve bilgisayar destekli eğitim alanına katkıda bulunacağı düşünülmüştür.

4. ÖNERİLER

Belirsiz limit problemlerinin çözümünde bir diğer yöntem olan sadeleştirme ve dönüştürme üzerinde çalışma yapılabilir.

Belirsiz limit ifadeleri üretiminde soru şablonlarını bir veritabanına kaydedilip o şablonlardan soru üretimi yapılacak şekilde bir çalışma yapılabilir.

Belirsiz limit ifadelerinin çözümü için yapay zekâ veya makine öğrenmesi yöntemlerinden faydalanılabilir. Bu sayede problemler için farklı çözüm stratejileri geliştirilebilir.

Bir görüntü içerisinden bulunan limit ifadelerinin görüntü işleme teknikleri ile metin formatına çevirilerek çalışmamızdaki sistemle adım adım çözümü gerçekleştirilebilir.

5. KAYNAKLAR

1. Cohen, J., S., Computer Algebra and Symbolic Computation:Mathematical Methods., A K Peters/CRC Press, 2003.

2. http://www.wolframalpha.com/ Wolframalpha. 16 Haziran 2016.

3. Bezuidenhout, J., Limits and Continuty: Some Conceptions of First-Year Students, International Journal of Mathematical Education in Science and Technology, 32, 4 (2001) 487-500.

4. https://javacc.dev.java.net/ JavaCC. 16 Haziran 2016.

5. Berlekamp, E. R., Factoring Polynomials Over Finite Fields, Bell System Technical Journal, 46, 8 (1967) 1853-1859.

6. https://en.wikipedia.org/wiki/Factorization_of_polynomials Factorization of Polynomials. 16 Haziran 2016.

7. Zassenhaus, H., On Hensel Factorization, I, Journal of Number Theory, 1, 3 (1969) 291-311.

8. Musser, D. R., Multivariate Polynomial Factorization, Journal of the ACM (JACM), 22, 2 (1975) 291-308.

9. Wang, P., S. ve Rothschild, L., P., Factoring Multivariate Polynomials Over the Integers, Mathematics of Computation, 29, 131 (1975) 935-950.

10. Risch, R. H., The Problem of Integration in Finite Terms, Transactions of the American Mathematical Society, 139, (1969) 167-189.

11. Nolan, J., F., Analytical Differentiation On a Digital Computer, Yüksek Lisans Tezi, Massachusetts Institute of Technology, 1953.

12. Kahrimanian, H. G., Analytical Differentiation by a Digital Computer, Yüksek Lisans Tezi, Temple University, 1953.

13. https://en.wikipedia.org/wiki/Lisp_(programming_language) Lisp (programming language). 16 Haziran 2016.

14. Slagle, J., R., A Heuristic Program That Solves Symbolic Integration Problems in Freshman Calculus, Journal of the ACM (JACM), 10, 4 (1963) 507-520.

15. Hearn, A., C., REDUCE: A User-oriented Interactive System For Algebraic Simplification., Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium., Washington, Bildiriler Kitabı, 79-90, Ağustos 1967.

16. Martin, W. A. ve Fateman, R. J., The MACSYMA System, SYMSAC '71 Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, Los Angeles, Bildiriler Kitabı, 59-75, Mart 1971.

17. Hearn, A., C., REDUCE 2: A System And Language For Algebraic Manipulation, Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation, Los Angeles, Bildiler Kitabı, 128-133, Mart 1971.

18. Griesmer, J., H. ve Jenks, R., D., SCRATCHPAD/1: An Interactive Facility For Symbolic Mathematics, Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, Los Angeles, Bildiriler Kitabı, 17-18, Mart 1971.

19. Wang, P., S., Automatic Computation of Limits, Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation, Los Angeles, Bildiriler Kitabı, 128-133, Mart 1971.

20. Ryan, C. ve O'Neill, M., Grammatical Evolution: Solving Trigonometric Identities, Proceedings of Mendel, 98, (1998)

21. Bauer, C., Frink, A. ve Kreckel, R., Introduction To The GiNaC Framework For Symbolic Computation Within The C++ Programming Language, Journal of Symbolic Computation, 33, 1 (2002) 1-12.

22. Carette, J., Understanding Expression Simplification, Proceedings of the International Symposium on Symbolic and Algebraic Computation, Santander, Bildiriler Kitabı, 72-79, Temmuz 2004.

23. Park, H., Symbolic Computation and Signal Processing, Journal of Symbolic Computation, 37, 2 (2004) 209-226.

24. Beeson, M. ve Wiedijk, F., The Meaning of Infinity in Calculus and Computer Algebra Systems, Journal of Symbolic Computation, 39, 5 (2005) 523-538.

25. Kredel, H., On a Java Computer Algebra System, Its Performance and Applications, Science of Computer Programming, 70, 2-3 (2008) 185-207.

26. Shatnawi, M. ve Youssef, A., Equivalence detection using parse-tree normalization for math search, 2nd International Conference on Digital Information Management ICDIM '07, Lyon, Bildiriler Kitabı, 643-648, Ekim 2007.

27. Lobachev, O. ve Loogen, R., Towards an Implementation of a Computer Algebra System in a Functional Language, International Conference on Intelligent Computer Mathematics, Birmingham, Bildiriler Kitabı, 141-154, Temmuz 2008.

28. Miyazaki, Y., Iguchi, Y. ve Watanabe, T., Information-Retrieval Tool for Math Expressions as Infrastructure of Training Engineers via E-Learning, 8th IFAC Symposium on Advances in Control Education, Kumamoto City, Bildiriler Kitabı, 302-306, Ekim 2009.

29. https://www.w3.org/Math/ MATHML. W3C, 16 Haziran 2016.

30. https://korpora-exp.zim.uni-duisburg-essen.de/naproche/ Naproche - Natural Language Proof Checking. 16 Haziran 2016.

31. Cramer, M., Koepke, P. ve Schröder, B., Parsing and Disambiguation of Symbolic Mathematics in The Naproche System, International Conference on Intelligent Computer Mathematics, Berlin, Bildiriler Kitabı, 180-195, Temmuz 2011.

32. Singh, R., Gulwani, S. ve Rajamani, S. K., Automatically Generating Algebra Problems, AAAI, (2012)

33. Fateman, R., Algorithm Differentiation in Lisp: ADIL, ACM Communications in Computer Algebra, 48, 3/4 (2015) 78-89.

34. Tekbaş, Y., Otomatik Kod Üretim Araçları Yardımıyla Matematiksel İfadelerin Türevlerinin Hesaplanması ve Sadeleştirilmesi, Yüksek Lisans Tezi, Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü, Trabzon, 2013.

35. Milani, M., Design and Applications of Grammar-based Methodologies for Automatic Generation and Step-by-step Solving of Mathematical Expressions, Doktora Tezi, K.T.Ü., Fen Bilimleri Enstitüsü, Trabzon, 2015.

36. Wolfram, S., Mathematica: A System For Doing Mathematics by Computer, ddison Wesley Longman Publishing Co., Inc., Redwood City, 1991.

37. http://www.maplesoft.com/ Maplesoft. 16 Haziran 2016.

38. http://www.mathworks.com/products/matlab/ Matlab. 16 Haziran 2016.

39. Adams, R., A., Calculus: A Complete Course, Pearson Education Limited, 2003.

40. Thomas, G., B., Calculus 1, Beta Basım Yayım Dağıtım A.Ş., İstanbul, 2001.

41. Hacısalihoğlu, H., H., Temel ve Genel Matematik, Bilecik, 2012.

42. Gamma, E. , Richard, H. , RalphJ. ve JohnV. , Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing Co., Inc, Boston, 1995.

43. https://maven.apache.org/ Apache Maven. 16 Haziran 2016.

44. http://junit.org/ Junit. 16 Haziran 2016.

45. http://www.oracle.com/technetwork/java/javaee/jsp/index.html JavaServer Pages Technology. 1 Mayıs 2016.

46. Appel, A., W., Modern Compiler Implementation in Java, Cambridge University Press, 2002.

47. Hacinliyan, A., Simgesel İşlem, Elektrik Mühendisliği, 338

ÖZGEÇMİŞ

Mehmet Cemil AYDOĞDU, 1985 Trabzon doğumludur. İlkokulu Mareşal Fevzi Çakmak İlk Okulu'nda, Ortaokul ve Liseyi ise Kanuni Anadolu Lisesi'nde tamamlamıştır. Anadolu Üniversitesi Bilgisayar Mühendisliği(İngilizce) Bölümü'nden 2010 yılında mezun olmuştur. 2013-2014 eğitim–öğretim yılının güz döneminde Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans programına, bahar döneminde de aynı bölümde araştırma görevlisi olarak çalışmaya başlamıştır. Halen Bilgisayar Mühendisliği Bölümü'nde Araştırma Görevlisi olarak çalışmaktadır. İyi derecede İngilizce bilmektedir.