KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

POLİNOMLAR İÇİN BİR SİMGESEL HESAPLAMA ÇATISININ TASARIMI VE GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Seda EFENDİOĞLU

HAZİRAN - 2017
TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

POLİNOMLAR İÇİN BİR SİMGESEL HESAPLAMA ÇATISININ TASARIMI VE GERÇEKLENMESİ

SEDA EFENDİOĞLU

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

Tezin Enstitüye Verildiği Tarih : 02 / 05 /2017
Tezin Savunma Tarihi : 01 / 06 /2017

Tez Danışmanı : Yrd. Doç. Dr. Hüseyin PEHLİVAN

Trabzon 2017

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Bilgisayar Mühendisliği Anabilim Dalında
Seda EFENDİOĞLU Tarafından Hazırlanan

POLİNOMLAR İÇİN BİR SİMGESEL HESAPLAMA ÇATISININ TASARIMI VE GERÇEKLENMESİ

başlıklı bu çalışma, Enstitü Yönetim Kurulunun 02 / 05 / 2017 gün ve 1700 sayılı kararıyla oluşturulan jüri tarafından yapılan sınavda
YÜKSEK LİSANS TEZİ
olarak kabul edilmiştir.

Jüri Üyeleri

Başkan : Prof. Dr. Vasif V. NABİYEV ……

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

Üye : Yrd. Doç. Dr. Rıfat BENVENİSTE ……

Prof. Dr. Sadettin KORKMAZ
Enstitü Müdürü

ÖNSÖZ

Fonksiyonların özel bir türü olan polinomlar, türevleri sürekli olduğu için mühendislik problemlerinin birçoğunun temsilinde ve çözümünde kullanılır. Simgesel hesaplama, bir problemin tam çözümünün bilgisayarlar yardımıyla bulunabilmesi için gereklidir. Bu yüzden mühendislik alanında yer alan polinom problemlerini iyi analiz edebilecek ve bu problemleri çözebilecek nitelikli simgesel hesaplama uygulamalarına ihtiyaç duyulur.

Yapılan tez çalışmasında, kullanıcıların polinom problemleriyle ilgili simgesel hesaplama işlemlerini desteklemek için kullanabilecekleri bir uygulama çatısının tasarım ve geliştirme aşamaları sunulmuştur. Ayrıca bu çatı üzerinde geliştirilmiş MathBox isimli uygulamanın, benzerlerini üretip adım adım çözebildiği polinom problemleri ve bu problemlerle ilişkili değerlendirmeler gösterilmiştir. Simgesel hesaplamayla çözülebilen polinom problemlerini temsil etmek için biçimsel dil tanımlamalarından yararlanılmıştır. Polinomların simgesel hesaplanması alanında iyi bir uygulama çatısının geliştirilebilmesi için literatürdeki çalışmalar incelenmiştir. Geliştirilen çatı, bu alandaki gereksinimleri karşılayacak niteliklere sahiptir.

Yüksek lisans tez çalışmamda danışmanlığımı üstlenen hocam Yrd. Doç. Dr. Hüseyin PEHLİVAN'a destek ve tecrübelerinden dolayı teşekkürlerimi borç bilirim. Her an yanımda olan, sevgi, ilgi ve bilgisini hiçbir zaman esirgemeyen eşim Hilmi Emre EFENDİOĞLU'na çok teşekkür ederim. Başta babam olmak üzere, dünyadaki en iyi baba olma ihtimali oldukça yüksektir, annem, ablam Selin, kızkardeşlerim Şeyma ve Gülsüm ile evimizin en küçüğü Miraç'ımıza sevgi dolu kalpleri, en stresli anlarımda bile bana karşı olan sabır ve anlayışları, kendimi en çaresiz hissettiğim zamanlarda bile bana verdikleri destek ve neşeden dolayı çok teşekkür ederim.

Seda EFENDİOĞLU

Trabzon 2017

TEZ ETİK BEYANNAMESİ

Yüksek Lisans Tezi olarak sunduğum "POLİNOMLAR İÇİN BİR SİMGESEL HESAPLAMA ÇATISININ TASARIMI VE GERÇEKLENMESİ" 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. 01/06/2017

Seda EFENDİOĞLU

İÇİNDEKİLER

Sayfa No

Yüksek Lisans Tezi

ÖZET

POLİNOMLAR İÇİN BİR SİMGESEL HESAPLAMA ÇATISININ TASARIMI VE GERÇEKLENMESİ

Seda EFENDİOĞLU

Karadeniz Teknik Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Yrd. Doç. Dr. Hüseyin PEHLİVAN
2017, 75 Sayfa

Simgesel hesaplama, bir problemin tam çözümünün bulunabilmesi için kullanılır. Simgesel hesaplamaları hatasız olarak hızlı bir şekilde yapabilen çok sayıda uygulama geliştirilmiştir. Ancak bu uygulamalar sadece kendi kullanıcı arayüzleri ile kullanılacak şekilde tasarlandığı için başka hesaplama ortamlarına entegre edilemezler veya bu ortamların bir bileşeni olarak çalıştırılamazlar.

Bu çalışmada, kullanıcıların polinomlarla ilgili simgesel hesaplama işlemlerini desteklemek için kullanabilecekleri bir uygulama çatısının tasarımı ve geliştirme aşamaları sunulmuştur. Bir polinom problemini temsil etmek için biçimsel dil tanımlamalarından yararlanılmıştır. Öncelikle polinom problemlerinin biçimlerine yönelik olarak gramer kuralları tanımlanmış ve dil ayrıştırma işlemleri uygulanarak ilgili problemlerin soyut sözdizim ağaçları üretilmiştir. Bu ağaçlar tüm programlama dilleri tarafından kullanılabilmeleri için JSON (JavaScript Object Notation) veri değişim biçimindeki ifadelere dönüştürülüp kaydedilmiştir. Ardından sadeleştirme işlemleri uygulanarak bu ifadelerin nesne temsilleri elde edilmiştir. Son olarak bu nesne temsilleri üzerinde çeşitli sayısal ve simgesel hesaplama işlemleri tanımlanarak polinom problemlerinin sadeleştirilmesine, adım adım çözülmesine ve benzerlerinin üretilmesine destek verilmiştir.

Anahtar Kelimeler: Sembolik Hesaplama, Polinomlar, Otomatik Problem Üretimi, Adım-Adım Problem Çözümü, Sadeleştirme, Gramerler, JavaCC, AST, JSON, UPA.

Master Thesis

SUMMARY

DESIGN AND IMPLEMENTATION OF A SYMBOLIC COMPUTATION FRAMEWORK FOR POLYNOMIALS

Seda EFENDİOĞLU

Karadeniz Technical University
The Graduate School of Natural and Applied Sciences
Computer Engineering Graduate Program
Supervisor: Asst. Prof. Dr. Hüseyin PEHLİVAN
2017, 75 Pages

Symbolic computation is used to find exact solution of a problem. Numerous applications have been developed that can quickly perform symbolic computations without errors. However, since these applications are designed to be used only with their own user interfaces, they can not be integrated into other computing environments or run as a component of those environments.

In this study, the development stages of an application framework that can be used by users to support symbolic computation operations on polynomials are presented. Formal language definitions have been used to represent a polynomial problem. Firstly, grammar rules are defined for the forms of polynomial problems, and abstract syntax trees of related problems are generated by applying language parsing operations. These trees are converted to expressions in JSON (JavaScript Object Notation) data interchange format so that they can be used by all programming languages. Then simplification operations are applied to obtain object representations of these expressions. Finally, various numerical and symbolic computation operations are defined on these object representations to support the simplification of polynomial problems, the step by step solution of problems and the generation of similar problems.

Key Words: Symbolic Computation, Polynomials, Automatic Problem Generation, Step-by-Step Problem Solution, Simplification, Grammars, JavaCC, AST, JSON, API.

ŞEKİLLER DİZİNİ

Sayfa No

TABLOLAR DİZİNİ

Sayfa No

KISALTMALAR DİZİNİ

BBG	Bağlamdan Bağımsız Gramer
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LL	Soldan Sağa Ayrıştırma - Soldan Türetim
LR	Soldan Sağa Ayrıştırma - Sağdan Türetim
UPA	Uygulama Programlama Arayüzü

1. GİRİŞ

Matematiksel hesaplamalar sayısal ve simgesel olmak üzere iki kısma ayrılır. Tarih boyunca el yordamıyla gerçekleştirilen sayısal hesaplamalar, elektronik cihazların gelişmesiyle birlikte hesap makinaları ve bilgisayar programları yardımıyla yapılmaya başlanmıştır. Ancak simgesel hesaplamaların elektronik ortama geçişi bilgisayarların ortaya çıkışıyla mümkün olabilmiştir. Bu geçişin en önemli aktörleri bilgisayarların yüksek hesaplama hızı ve sembolik işlem yapabilme yeteneğidir. [1] Simgesel hesaplama yapabilen bilgisayar uygulamalarının ilk örnekleri 1960'larda geliştirilmeye başlanmıştır. Bu uygulamalarda, sayısal eşitlikler yerine simgesel eşitliklerden yararlanılarak, simgeler üzerinden hesaplama yapılır. Bu nedenle, sayısal hesaplamalar yaklaşık değerler üretirken, simgesel hesaplamalar bir problemin tam çözümü ile ilgilenir. [2]

Polinomlar, mühendislik problemleri için geliştirilen matematiksel modellerin önemli bir bileşenini meydana getirmektedir. [3] Polinomlar, bilgisayar grafikleri [4], sır paylaşımı [5], kriptografi [6], robotların konum ve hareket kontrolü [7], sayısal sinyal işleme [8] gibi birçok farklı mühendislik probleminin temsil edilmesinde ve çözümünde kullanılmaktadır.

Problemlerin otomatik üretilmesi, sadece bir problemi temsil eden matematiksel modelin belirlenmesinde değil, matematiğin öğretiminde de büyük rol oynamaktadır. Bir problem türünün çözümünde zorluk yaşayan bir öğrenci, üretilen benzer problemleri çözerek, kendisini ilgili alanda geliştirebilmektedir.

Bir problemin benzerini otomatik olarak üretme, 1960'lı yıllardan beri çalışılan bir alandır. [9] Ancak son yıllarda bu alanda yapılan çalışmalar nitelik ve nicelik bakımından oldukça artmıştır. Bir problemin benzerlerinin otomatik üretildiği modellerin alanları; polinom problemlerinin çözümünü ve sadeleştirilmesini yapabilen bilgisayar programı üretme [10], sırt çantası problemi üretme [11], bir bilgisayar programının benzerini üretme [12], [13], bilgisayar programlarını otomatik test edebilecek test kodu üretme [14], gömülü sistemlere ait problem üretme [15], programlama öğrenenlere yardımcı olmak için otomatik ipucu üretme [16], mantık problemi üretme [17], bilgisayar mimarisi problemi üretme [18], otomata problemi üretme [19], otomatik olarak bağlamdan bağımsız gramer üretme [20], [21] gibi birçok farklı alana yayılmıştır.

Özellikle matematik alanında yapılan çalışmaların sayısı son yıllarda farkedilir şekilde artmıştır. Matematiğin öğretiminde, bir problemin çözümü kadar o problemin benzerlerinin üretilmesinin de önemli olduğu düşünülmektedir. [22] Bu nedenle matematik alanında, denklem sistemlerini çözebilen bir sistem üretme [23], temel matematik problemlerini öğretebilmek için basit bir oyun üretme [24], geometri problemi üretme [25], metinsel bilgiler içeren matematik problemlerini otomatik üretme [26] gibi farklı problem üretme çalışmaları yapılmıştır.

Özel olarak, metinsel veri içermeyen matematik problemlerinin benzerlerinin üretilmesi alanında yapılan çalışmalara bakıldığında, farklı yaklaşımlar görülmektedir. Jurkovic'in çalışmasında, kullanıcılardan alınan ya da önceden belirlenen bir şablona uygun matematik problemlerinin benzerlerini üreten bir uygulama sunulmaktadır. [27] Şablonda yer alan değişkenlerin alabilecekleri en küçük ve en büyük değerler, benzer ifade üretiminden önce kullanıcıdan alınmaktadır. Üretim esnasında sadece bu değerler değiştirilmektedir. Bu yüzden, bu uygulama yardımıyla üretilen problemlerin, sadece sayısal bileşenleri farklı olabilmektedir. Singh ve arkadaşları sundukları çalışmada, kendi geliştirdikleri sorgu dili yardımıyla matematiğin polinom, seri toplamı gibi birçok farklı alt alanına ait problemlerin benzerlerini üretebilen bir uygulamadan bahsetmektedir. [28] Bu uygulamada, kullanıcılardan benzerlerini üretmek için alınan problemlerin, eşitlik biçiminde olma zorunluluğu vardır.

Matematik problemlerinin bilgisayar yardımıyla çözümü 1950'li yıllardan beri çalışılan bir alandır. [29], [30] Ancak bu çalışmalarda geliştirilen uygulamalar, bir problemin çözüm adımlarını göstermeden, doğrudan sonucu sunmaktadır. Kara kutu olarak adlandırılan bu yaklaşım nedeniyle, bu uygulamalar matematiğin öğretimine destek verecek niteliklere sahip değildi. Bu uygulamalara, bir problemin çözüm aşamalarının açıkca gösterilmesi olan cam kutu yaklaşımının dahil edilmesiyle, bu eksikliğin giderilebileceği düşünülmüştür. [31] Bu düşünceyi dikkate alarak geliştirilen uygulamalardan biri Beeson tarafından geliştirilen Mathpert [32], diğeri de Jurkovic tarafından geliştirilen Algebrator [27]'dür. Bahsedilen uygulamalar, ilgilendikleri problem uzayları sınırlı olduğu için yeterince başarılı olamamışlardır. [33], [34] Açık kaynaklı olarak, bir grup gönüllü programcı tarafından geliştirilen SymPy'nin ilgilendiği problem uzayı oldukça geniştir. [35] Ancak SymPy çevrimiçi ortamda sunucu tarafında çalıştığı için, kullanıcı kaynaklarından faydalanılamamaktadır. [36] 1960'lı yılların sonunda Massachusetts Institute of Technology (MIT) tarafından geliştirilen ve ilk bilgisayarlı cebir sistemlerinden biri olan Macsyma, 1990'lı yılların sonundan itibaren açık

kaynaklı olarak Maxima adıyla geliştirilmeye devam etmektedir. [37] Maxima masaüstü uygulaması olarak başarılı olsa da, çevrimiçi kullanıma uygun değildir. 1980'li yılların sonunda Wolfram Research tarafından geliştirilen Mathematica, ticari anlamda başarıya ulaşan ilk bilgisayarlı cebir sistemidir. [38] Günümüzde en çok kullanılan simgesel hesaplama uygulamalarından biri olan Mathematica, ücretsiz olarak kullanılamamaktadır.

Bilgisayarlar yardımıyla problem çözmenin yaygınlaşmasıyla birlikte, bilgisayarların bir problemin çözümünü daha hızlı ve daha doğru nasıl üretebileceği düşünülmüş ve bu amaçla çalışmalar yapılmıştır. [39], [40] Yapılan çalışmalar günümüzde de devam etmektedir. [41], [42] Bu amaçla yapılan sadeleştirme işlemlerine, sadece çözüm aşamalarında değil üretim aşamalarında da ihtiyaç duyulabilmektedir. Çünkü, kullanıcılar tarafından sisteme girilen problemler, her zaman açık ve anlaşılır olmayabilir. Alınan bir problemin benzerlerinin üretilebilmesi için, problemin öncelikle sadeleştirilmesi gerekir. Var olan simgesel hesaplama uygulamalarında tanımlı sadeleştirme işlemleri yeterli olmamaktadır. Sadeleştirme alanında yapılan çalışmalar genellikle bu uygulamalar üzerinde geliştirilmektedir. Bu çalışmalardan birinde Bailey ve arkadaşları yaygın kullanılan simgesel hesaplama uygulamalarından birinin temelini kullanarak, serilerin toplamının sadeleştirilmesini sağlayan bir yazılım paketi geliştirmiştir. [43]

Simgesel hesaplama yapabilen kapsamlı uygulamalar, içerisinde birçok küçük uygulamayı barındırır. Bu küçük uygulamaların bazıları, simgesel hesaplama yapabilen uygulamaların temel yapısını oluşturduğu için, kesinlikle geliştirilen uygulamaya eklenmelidir. Özel bir alanda simgesel hesaplama uygulaması geliştirmeye çalışan bir programcı, öncelikle bu temel yapıyı oluşturmak zorundadır. Bu nedenle aynı temel yapı, farklı geliştiriciler tarafından farklı uygulamalar için, defalarca yeniden yazılmaktadır. Bu durum hem geliştiricilerin vakit kaybetmesine hem de simgesel hesaplama yapabilen uygulamaların, farklı alanlardaki gelişiminin gecikmesine neden olmaktadır. Bu soruna çözüm olarak simgesel hesaplama yapabilen ugulama çatıları ortaya çıkmıştır. Bu çatılardan biri de Bauer ve arkadaşları tarafından geliştirilen GiNAC'dır. [44] GiNAC, mevcut bilgisayarlı cebir sistemlerinin temel yapısı geliştiricilerin kendi uygulamalarında kullanılmaya uygun olmadığı için, geliştiricilerin üzerinde kendi uygulamalarını geliştirebilecekleri bir simgesel hesaplama çatısı olarak önerilmiştir. [45]

Tez kapsamında, mevcut bilgisayarlı cebir sistemleri ve simgesel hesaplama yapabilen uygulama çatıları incelenerek, tespit edilen eksikliklerin giderilmeye çalışıldığı, özellikle polinomlar için bir simgesel hesaplama çatısı geliştirilmiştir. Bu çatı ilk olarak metin formatındaki polinom problemlerini ayrıştırıp değerlendirir. Ardından ortaya çıkan ifadeye sadeleştirme işlemleri uygulayarak benzer problemlerin üretilmesini ve problemlerin adım adım çözümünün yapılmasını kolaylaştırır. Son olarak sade şekildeki problemlere benzer yeni problemler üretir veya bu problemleri adım adım çözer. Ayrıca geliştirilen çatıya, diğer geliştiricilerin kendi uygulamalarına entegre edebilmeleri için, hem çevrimiçi hem de çevrimdışı çalışabilecek bir uygulama programlama arayüzü (UPA) eklenmiştir. Uygulama çatısının bileşenleri sadece sunucu tarafında değil istemci tarafında da çalışabilecek şekilde geliştirildiği için, simgesel hesaplama işlemleri sırasında sunucunun yükü kullanıcı bilgisayarlarıyla paylaşılmaktadır. Geliştiriciler tasarlanan uygulama çatısını kendi uygulamalarına kolaylıkla entegre edebilecektir. Böylece geliştiriciler kendi uygulamalarına kısa sürede simgesel işlem yapabilme kabiliyeti kazandırarak, vakit kaybetmeden geliştirmek istedikleri uygulamalarının özel işlevlerine odaklanabilecektir. Ayrıca kullanıcılar bu çatı yardımıyla geliştirilen uygulamaları kullanarak, problemlerini temsil eden polinomların benzerlerini üretebilecek, problemlerini adım adım çözebilecek ve böylece alanında yetkin hale gelebilecektir.

2. GENEL BİLGİLER

2.1. Polinomlar

A kümesi tanım kümesi ve B kümesi değer kümesi olmak üzere tanımlanmış $f(x)$ fonksiyonu $f : A \rightarrow B$ şeklinde gösterilir. Tanım kümesinde yalnızca doğal sayılar olan fonksiyonlara polinomlar denir.

Polinomlar, $a_0, a_1, \ldots, a_n$ (katsayılar) eleman R(reel sayı kümesi) ve n (dereceler) eleman N (doğal sayı kümesi) olmak üzere $P(x) = a_n x^n + ... + a_1 x + a_0$ şeklinde gösterilir.

2.1.1. Çalışma Kapsamında İlgilenilen Polinom Problemleri

Geliştirilen uygulama çatısı kullanılarak simgesel hesaplama alanında farklı problem uzaylarıyla ilgilenen uygulamalar geliştirmek mümkündür. Bu çalışma kapsamında, kullanıcıdan alınan polinom sorularının benzerlerini otomatik olarak üretebilen ve adım adım çözebilen uygulama çatısı kullanılarak, MathBox isimli bir uygulama geliştirilmiştir. Bu çatının ilgilendiği polinom problemleri genelleştirilerek sekiz türde temsil edilebilir. Bu türler ve isimleri Tablo 2.1'de, her bir türe ait bazı problem örnekleri Tablo 2.2'de gösterilmiştir.

Tablo 2.1. Problem türleri

Tür	İsim
1	Verilen Polinomun Başka Bir Değer için Karşılığını Bulma
2	Bilinmeyeni Bulma
3	Derece Bulma
4	Aritmetik İşlem Problemleri
5	Derece : Aritmetik İşlem Problemleri
6	Bileşke Fonksiyon Bulma
7	Eşitlik Problemleri
8	Binom Açılımı Yapma

Tablo 2.2. Problem örnekleri

Tür	Örnek
1	$P(x-1) = x^3 - x + 2 \quad P(0) + P(1) = ?$ $P(x) = 3x + 2 \quad P(x+2) = ?$
2	$P(x) = ax^2 + 3x + 7 \quad P(2) = 11 \quad a = ?$
3	$P(x) = 4x^6 + 3x^2 - 6x + 5 \quad der[P(x)] = ?$ $P(x) = x^2 + x - 1 \quad der[P(x^2)] = ?$
4	$P(x) = 2x + 6 \quad Q(x) = 3x + 1 \quad P(x) + Q(x) = ?$ $P(x) = 2x + 6 \quad Q(x) = 3x + 1 \quad P(x) - Q(x) = ?$ $P(x) = 2x + 6 \quad Q(x) = 3x + 1 \quad P(x) * Q(x) = ?$ $P(x) = 27x^4 - 6x^2 + x + 15 \quad Q(x) = 3x - 1 \quad P(x)/Q(x) = ?$
5	$P(x) = 2x + 6 \quad Q(x) = 3x + 1 \quad der[P(x) + Q(x)] = ?$ $P(x) = 2x + 6 \quad Q(x) = 3x + 1 \quad der[P(x) - Q(x)] = ?$ $P(x) = 2x + 6 \quad Q(x) = 3x + 1 \quad der[P(x) * Q(x)] = ?$ $P(x) = 27x^4 - 6x^2 + x + 15 \quad Q(x) = 3x - 1 \quad der[P(x)/Q(x)] = ?$
6	$P(x) = x^2 + x \quad Q(x) = x^2 \quad P(Q(x)) = ?$
7	$P(x) = (a-1)x^3 - bx^2 + 3x + 1 \quad Q(x) = x^2 + cx + d \quad P(x) = Q(x)$ $a + b + c + d = ?$
8	$P(x) = x^2 + 3x + 1 \quad P(x))^4 = ?$

2.1.2. Polinomların Bilgisayar Ortamında Temsil Edilmesi

Polinomlar metin, dizi ve ağaç gibi bir çok veri yapısı ile temsil edilebilir.

Metin:

Polinomlar `P(x)=x^3+3x^2+2x+2, Q(x)=x^2+3x+1` gibi basit metin formatında temsil edilebilir. Bu temsil polinomun adını ve farklı dereceler için bilinmeyenlerin katsayılarını içermektedir.

Dizi:

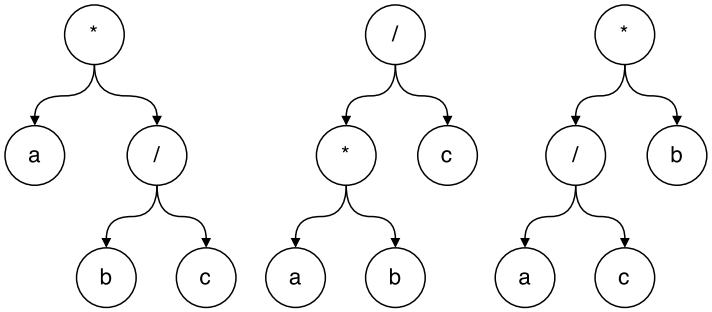
Bir polinomun adına gerek duyulmuyorsa ve bilinmeyenlerinin dereceleri sıralı ilerliyorsa, o polinom bir dizi içerisindeki katsayılarla temsil edilebilir.

Örneğin metin formatında `P(x)=x^3+3x^2+2x+2` şeklinde temsil edilen bir polinom, dizi formatında $[1, 3, 2, 2]$ şeklinde temsil edilebilir.

Ağaç:

Ağaç veri yapıları polinomların temsilinde kullanılabilecek en uygun veri yapılarıdır. Bu durumun temel nedenleri, ağaçların polinomlara ait bütün verileri kayıpsız tutabilmesi, esnek olması ve hiyerarşik bir yapı sunmasıdır.

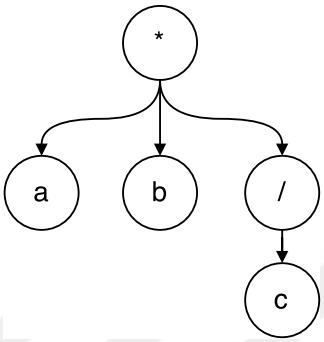
Bazı ifadeleri temsil eden birden fazla soyut sözdizim ağacı olabilir. Örneğin, Şekil 2.1'de gösterildiği gibi, ab/c ifadesini temsil edebilecek üç farklı soyut sözdizim ağacı vardır.



Şekil 2.1. ab/c ifadesini temsil eden ağaçlar

Aynı matematiksel ifadenin farklı soyut sözdizim ağaçlarıyla temsil edilebilmesi, daha sonraki işlem aşamalarında farklı sorunlara sebep olabilir. Örneğin, her bir farklı soyut sözdizim ağacı farklı sadeleştirme işlemlerine ihtiyaç duyabilir. Bu durum simgesel hesaplama yapacak uygulamalar için bir dezavantajdır.

Yapılan çalışmada bu durumun oluşmasını engellemek için özel bir soyut sözdizim ağacı yapısı önerilmiştir. Toplama ve çarpma gibi temel matematiksel işleçlerin çocuk düğüm sayısı sabit tutulmamış, ifadede yer alan işlenen sayısına göre dinamik olarak değişecek şekilde ayarlanmıştır. Bu yaklaşımla, ab/c ifadesi Şekil 2.2'de gösterildiği gibi sadece bir soyut sözdizim ağacıyla temsil edilebilecek hale gelir.



Şekil 2.2. Önerilen ağaç

2.2. Ağaçlarda Dolaşma Yöntemleri

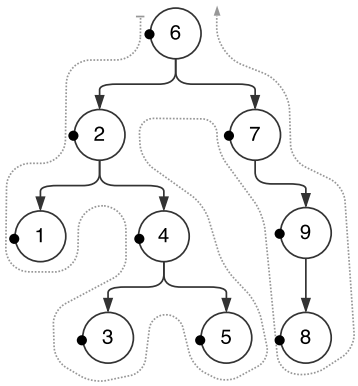
Bilgisayar bilimlerinde ağaçlarda dolaşma, bir graf gezme biçimidir ve bir ağaç veri yapısındaki her düğümü kontrol etme, güncelleştirme gibi amaçlar için ziyaret etme işleminin adıdır. Ağaç gezinme yöntemleri, düğümlerin ziyaret sırasına göre sınıflandırılır.

2.2.1. Kök Önce Yöntemi

Kök önce yönteminin algoritması aşağıdaki gibidir:

1. Geçerli düğümün boş olup olmadığı kontrol edilir.
2. Kökün (veya geçerli düğümün) veri bölümü ziyaret edilir.
3. Soldaki alt ağaç için kök önce arama işlemi özyinelemeli olarak çağrılır.
4. Sağdaki alt ağaç için kök önce arama işlemi özyinelemeli olarak çağrılır.

Örnek bir ağacın kök önce yöntemiyle dolaşılması Şekil 2.3'de gösterilmiştir.



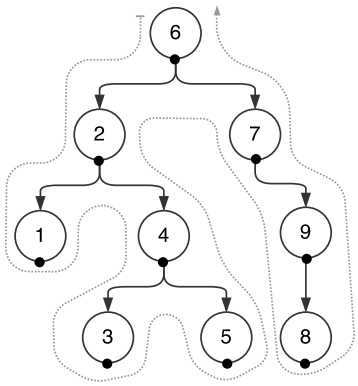
Şekil 2.3. Kök önce: 6, 2,
1, 4, 3, 5, 7, 9, 8.

2.2.2. Kök Ortada Yöntemi

Kök ortada yönteminin algoritması aşağıdaki gibidir:

1. Geçerli düğümün boş olup olmadığı kontrol edilir.
2. Soldaki alt ağaç için kök ortada arama işlemi özyinelemeli olarak çağrılır.
3. Kökün (veya geçerli düğümün) veri bölümü ziyaret edilir.
4. Sağdaki alt ağaç için kök ortada arama işlemi özyinelemeli olarak çağrılır.

Örnek bir ağacın kök ortada yöntemiyle dolaşılması Şekil 2.4'de gösterilmiştir.



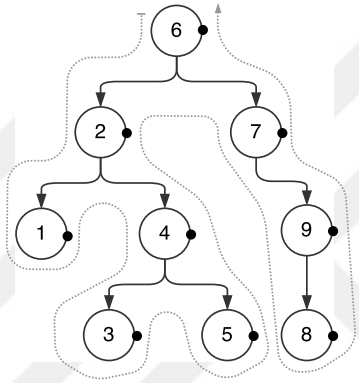
Şekil 2.4. Kök ortada: 1, 2,
3, 4, 5, 6, 7, 8, 9.

2.2.3. Kök Sonda Yöntemi

Kök sonda yönteminin algoritması aşağıdaki gibidir:

1. Geçerli düğümün boş olup olmadığı kontrol edilir.
2. Soldaki alt ağaç için kök ortada arama işlemi özyinelemeli olarak çağrılır.
3. Sağdaki alt ağaç için kök ortada arama işlemi özyinelemeli olarak çağrılır.
4. Kökün (veya geçerli düğümün) veri bölümü ziyaret edilir.

Örnek bir ağacın kök sonda yöntemiyle dolaşılması Şekil 2.5'de gösterilmiştir.



Şekil 2.5. Kök sonda: 1, 3, 5, 4, 2, 8, 9, 7, 6.

Çalışmada üretilen soyut sözdizimi ağaçları kök sonda yöntemiyle dolaşılmaktadır. Bunun nedeni en sade haldeki çocuk düğümlerle işlem yapılmak istenmesidir. Ayrıca çalışmadaki soyut sözdizim ağaçları iki çocuk düğümle sınırlı olmadığından, kök sonda yönteminin algoritması, değişiklik yapıldıktan sonra kodlanmıştır. Algoritmanın ikinci adımıyla üçüncü adımı arasına, en sol düğümden en sağ düğüme gelene kadar özyinelemeli çağrımı sağlayacak ifadeler eklenmiştir. Bu sayede bir kök düğüme bağlı birden fazla çocuk düğüm dolaşılabilir.

2.3. Çeviriciler

2.3.1. Yorumlayıcılar

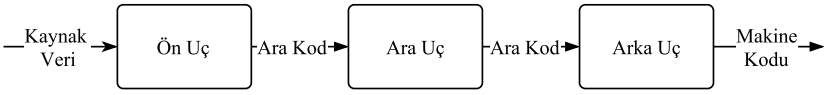
Yorumlayıcılar, kullanıcıdan bir arayüz yardımıyla alınan veya bir dosyadan okunan kaynak veriyi bloklar halinde veya satır satır okuyarak çalıştırır. Bu yüzden yorumlayıcılar çalıştırılabilir bir kod dosyası üretmezler. Yorumlama işlemi satır satır yapıldığı için, ilk yorumlanamayan satırda hata verilir ve programın çalışması kesilir. Programda hatalı olan fakat yorumlayıcının çalışması sırasında ilgilenilmeyen satırlar, yorumlayıcı için bir sorun oluşturmaz.

Yorumlayıcılar kaynak koddan makina diline doğrudan çeviri yaptıkları için, kaynak kodun her çalıştırılma sürecinde yeniden çeviri yapılır. Bu yüzden yorumlayıcılar yavaş çalışan çeviricilerdir ve yorumlayıcı tarafından çevrilecek kaynak kodun iyileştirilme imkanı sınırlıdır. Ancak bu durum kaynak kodda yapılan değişikliğe alınan cevap süresini olumlu etkilemektedir. Kaynak kod satır satır çevrildiği için, hem kaynak kodda değişiklik yapmak kolaydır hem de yapılan değişikliğin etkisini görmek hızlıdır.

2.3.2. Derleyiciler

Derleyiciler ön uç , ara uç ve arka uç olmak üzere üç ana kısımdan oluşur. Ön uç, kullanıcıdan bir arayüz yardımıyla alınan veya bir dosyadan okunan kaynak verinin analizinin hedef makinadan bağımsız olarak yapıldığı yerdir. Analiz sonucunda oluşan ara kod, ara uca iletilir. Programın performansının iyileşmesi için, ara uç tarafından ara koda bazı optimizasyonlar yapılır. Elde edilen optimum ara kod, arka uç tarafından hedef makinanın mimarisi dikkate alınarak makina koduna dönüştürülür. Tüm bu aşamalar tamamlandığında, kaynak veriden ilgili makinada çalıştırılmaya hazır makina kodu elde edilmiş olur.

Temel derleyici aşamaları Şekil 2.6'da gösterilmiştir.

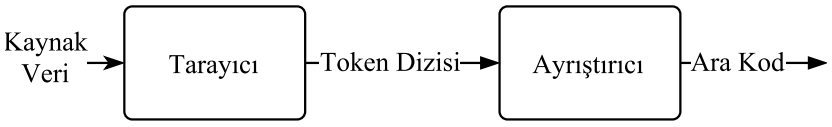


Şekil 2.6. Derleyici aşamaları

2.3.2.1. Ön Uç Aşamaları

Ön uç aşamaları, kaynak verinin komut satırı veya dosyadan okunmasıyla başlayıp ara kodun üretilmesiyle son bulur. Bu ara kod bir soyut sözdizim ağacıdır.

Ön uç aşamaları Şekil 2.7'de gösterilmiştir.



Şekil 2.7. Ön uç aşamaları

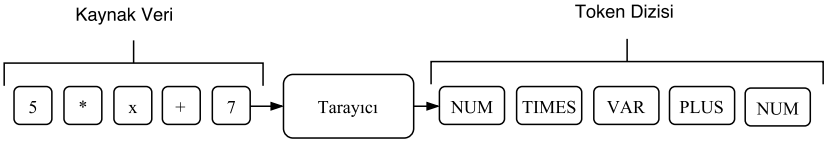
2.3.2.1.1. Sözcüksel Analiz

Kullanıcıdan komut satırı veya arayüz yardımıyla alınan veya dosyadan okunan kaynak veri ilk olarak sözcüksel analizden geçer. Kaynak veri, geliştiriciler tarafından önceden tanımlanmış, token adı verilen sözcüklere göre ayrıştırılır. Sözcüksel analiz aşamasında bir hata oluşmazsa kaynak veri bir token dizisine dönüştürülür. Kaynak veriden okunan bir ifade için token üretilememesi durumunda sözcüksel hata verilir ve kullanıcıdan girdiği ifadeyi gözden geçirmesi beklenir.

Temel sözcüksel analiz algoritması aşağıdaki gibidir:

1. Veriyi okumaya başla
2. token= sonrakiToken( );
3. Token için işlemler yap
4. Veri bitene kadar 2. adıma dön

Örnek bir kaynak veri ve bu veri için tarayıcı tarafından üretilen token dizisi Şekil 2.8'de gösterilmiştir.



Şekil 2.8. Örnek bir sözcüksel analiz

2.3.2.1.1.1. Düzenli İfadeler ve Token Tanımlama

Düzenli ifadeler, bilgisayar biliminde ve dil teorisinde bir arama kalıbını tanımlayan bir dizi karakterdir. Genellikle bu kalıplar metin arama algoritmalarında metin bulma, metni bulup değiştirme gibi işlemler için kullanılır.

Token tanımlaması yapılırken düzenli ifadeler kullanılır. Çünkü düzenli ifadeler, sonlu sayıda ifadeyle sonsuz sayıda ifadeyi temsil edebileceğimiz yapılardır. Doğal dilde her bir karakter dizisi anlamlı olamayacağı için doğal dili analiz etmek zordur. Oysa programlama dillerinde düzenli ifadeler sayesinde tüm karakter dizilerini analiz etmek mümkündür.

2.3.2.1.1.2. Token Tanımlarken Dikkat Edilmesi Gerekenler

Derleyici tasarlarken tokenların doğru tanımlanması çok önemlidir çünkü token tanımlamalarının düzgün yapılmadığı durumlarda alınan kaynak veri için doğru bir token dizisi üretilemeyecektir. Yanlış bir token dizisiyle derleyicinin diğer aşamaları yürütülemeyecektir. Bu durumun önüne geçmek için, kaynak veriden okunan her bir karakter dizisi için yalnızca bir ve doğru tokenı üretecek bir gramerin yazılması gerekir.

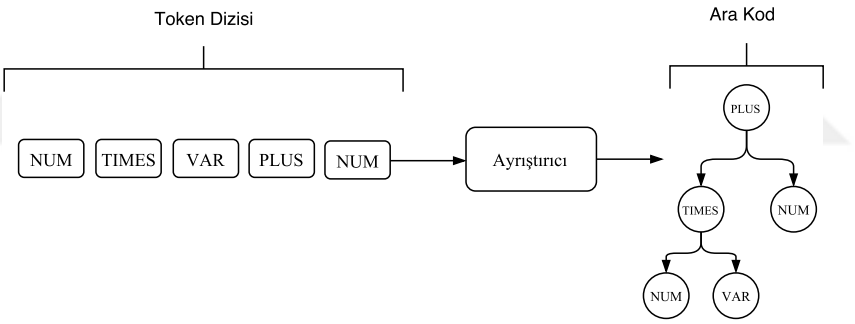
Burada karşımıza iki soru çıkar. Bunlardan ilki, kaynak veriden okunan kaç karakter için bir token üretilsin sorusudur. Bu durum en büyük eşleme olarak adlandırılır. Sözcüksel analizci (tarayıcı) her zaman en fazla sayıda karakterin oluşturduğu bir karakter dizisi için bir token üretmeye çalışır. Örneğin i, f ve if girdileri için sözcüksel analizcinin i'yi okuyup ardından f'yi okuduğunda sırasıyla i ve f'ye karşılık gelen tokenları üretmesi beklenir. Fakat sözcüksel analizci herzaman if'e karşılık gelen tokenı üretir. Bu durum sözcüksel hataya sebep olmaz fakat elde edilen token dizisini inceleyen sözdizimsel analizci bir sonraki aşamada sözdizimi hatası verir.

Diğer soru da, kaynak veriden okunan bir karakter dizisi aynı anda iki tokenla eşlenebiliyorsa, bu karakter dizisini hangi tokenla eşleriz sorusudur. Bu durumda ilk karşılaşılan tokenla eşleme yapılır. Örneğin yazılan gramerde isimleri temsil eden bir NAME tokenı ve anahtar kelimeleri temsil eden bir KEYWORD tokenı olsun. Tanımlanan gramerde, "var" karakter dizisi bu iki token tarafından da eşlenebiliyor olsun. Eğer KEYWORD tokenı gramerde NAME tokenından önceyse, sözcüksel analizci kaynak veriden "var" karakter dizisini her okuduğunda karşılığında KEYWORD tokenını üretecektir.

2.3.2.1.2. Sözdizimsel Analiz

Sözdizimsel analizci (ayrıştırıcı), kaynak veri olarak tarayıcıdan aldığı token dizisinin önceden tanımlanan sözdizimi kurallarına uygun olup olmadığını kontrol eder ve bir sorun görmezse soyut sözdizim ağacını oluşturur. Bazen okunan kaynak veri için farklı soyut sözdizim ağaçları üretilebilir. Bu duruma belirsiz gramerler sebep olur. Belirsiz gramerlerde oluşan ağaç sola veya sağa doğru büyüyebilmektedir. Oysa JavaCC derleyici üretici aracının, ayrıştırıcısını üreteceği gramer soldan özyinelemeli olamaz. Analiz sonucu üretilen ağaç, sağa doğru büyüyen bir ağaç olmalıdır. Ancak gramer yazılırken sadece soldan özyinelemeliliği engellemek belirsizliği ortadan kaldırmada yeterli olmaz. Aynı zamanda işleçlerin işlem önceliklerine ve birleşme yönlerine de yazılan gramerde dikkat edilmelidir.

Soyut sözdizim ağaçları genellikle ön uç ve arka uç arasında ara kod olarak kullanılırlar. Örnek bir token dizisi ve bu token dizisi için ayrıştırıcı tarafından üretilen ara kod Şekil 2.9'da gösterilmiştir.



Şekil 2.9. Örnek bir sözdizimsel analiz

2.3.2.1.2.1. LL Ayrıştırma

İsimde yer alan ilk L, okunan girdi verisinin soldan sağa doğru tokenlarına ayrıştırılacağını belirtir. İkinci L ise, sözcüksel analizciden alınan token dizisinden, sözdizim kuralı türetiminin soldan sağa yapılacağını belirtir. Yukarıdan aşağıya ayrıştırma olarak da bilinir.

2.3.2.1.2.2. LR Ayrıştırma

İsimde yer alan L, okunan girdi verisinin soldan sağa doğru tokenlarına ayrıştırılacağını belirtir. R ise, sözcüksel analizciden alınan token dizisinden sözdizim kuralı türetiminin sağdan sola yapılacağını belirtir. Aşağıdan yukarıya ayrıştırma olarak da bilinir.

2.3.2.2. Orta Uç Aşaması

Ön uç ile arka uç arasında ve tüm arka uç aşamalarında çalışır. Programın çalışma zamanını iyileştirir. Ölü kod eliminasyonu, gereksiz bellek alanlarının boşaltılması, ortak ifadelerin sadeleştirilmesi gibi birçok farklı optimizasyon yapar. Kaynak kodun ara koda dönüştürülme aşamasında sadece bir kere değil defalarca çalışarak optimum çalıştırılabilir kodun elde edilmesine yardım eder.

2.3.2.3. Arka Uç Aşamaları

Arka uç aşamalarından olan emir seçimi aşamasında, ara koddaki kalıplara karşılık en yüksek performansı sağlayacak emirler (assembly dilinde) üretilir. Bu işlem hedef makina mimarisine bağlı yapılır. Bir veya birden fazla düğüm gruplanarak bu grup için bir emir üretilir. Ara kod olarak kullanılan sözdizim ağacı için, optimum sayıda emri üretmek zor bir problemdir. Sonuçta üretilen emir dizisi kaynak kod olarak birleştiriciye (assembler) yollanır. Yüksek seviyeli bir dilde çalıştırılabilir bir kod üretilemeyeceğinden, birleştirici kaynak kodu makina dilindeki bir çalıştırılabilir koda dönüştürür.

Arka uç aşamalarından bir diğeri olan kaydedici tahsisi aşamasında, hangi verinin kaydedicide hangi verinin bellekte tutulacağına karar verilir. Kaydedici alanı sınırlı olduğu için, değişkenler arasında değerlendirme yapılarak sadece icra esnasında bulunması gereken değişkenler kaydedicide, diğerleri bellekte tutulur. Ayrıca değişkenlerin birbiriyle örtüşme durumları kontrol edilerek, birbiriyle örtüşmeyen değişkenler için, aynı zamanda aynı kaydedici tahsis edilir.

2.3.2.4. Derleyici Üreten Araçlar

Yazılan gramere uygun sözcüksel analizci ve sözdizimsel analizciyi üretirler. Lex, Bison, JavaCC gibi farklı programlama dilleri veya platformlar için geliştirilmiş örnekleri vardır. Yapılan çalışmada Java programlama dilinde kod üretebilen JavaCC derleyici üretici aracı kullanılmıştır.

2.3.2.4.1. JavaCC

JavaCC, Java dilinde yazılan bir derleyici üretici aracıdır. [47] JavaCC Specification File isimli dosyaya, LL formunda yazılan gramerler için bir derleyici üretir. Girdi verisini soldan sağa doğru okuyarak tokenlarına ayıran bir sözcüksel analizci ve oluşan token dizisi için soyut sözdizim ağacını soldan sağa doğru oluşturan bir sözdizimsel analizci üretir. Soldan özyinelemeli gramerler için derleyici üretemez. Bu dosyada, kaynak veride bulanan ama karşılığında token üretmek istemediğimiz karakter dizileri için SKIP tokenı tanımlanır ve bu sayede satır sonu karakter dizisi, yorum satırları gibi girdi verileri için JavaCC'nin token üretmesi engellenir. Bu tür karakter dizilerinin, sözcüksel analiz aşamasından önce kaynak veriden atılmamasının sebebi, sözcüksel analizin doğru biçimde yapılabilmesini garanti etmektir.

2.4. Uygulama Çatıları

Uygulama çatıları, yeni geliştirilen bir yazılım platformunun parçası olarak kullanılarak ilgili yazılıma kendine has işlevsel özelliklerini katar. Uygulama çatıları, tekrar tekrar kullanılabilen ve evrensel standartlara sahip yazılım ortamlarıdır.

Uygulama çatıları, bir projenin veya bir sistemin geliştirilmesini sağlamak için farklı bileşenleri bir araya getirir. Bu bileşenler destek programları, derleyiciler, kod kütüphaneleri, araç setleri ve uygulama programlama arayüzleri (UPA) olabilir. Bu bileşenler uygulama geliştiricilerine, zaten var olan temel işlevleri tekrar kendileri sağlamak yerine, kendi problemlerine odaklanma imkanı sağlar. Bu sayede, bir uygulama çatısının desteklediği alanda geliştirilen uygulamaların nitelik ve nicelik bakımından artışı sağlanmış olur.

2.5. Kullanılan Teknolojiler

2.5.1. JavaScript

JavaScript dili, üst seviye, dinamik ve yorumlanan bir programlama dilidir. Web sitelerinin çoğunluğu JavaScript kullanır ve tüm modern web tarayıcıları ek yazılımlara ihtiyaç duymadan JavaScript ile yazılmış uygulamaları çalıştırabilir. Yeni gelişmeler sonucunda JavaScript dili, tarayıcılar dışında çalışabilen uygulamalar ierisinde kullanılabilecek olanaklara sahip olmuştur.

Bu dilde fonksiyonlar başka bir fonksiyona argüman olarak verilebilir, bir fonksiyon başka bir fonksiyona sonuç olarak döndürülebilir, bir fonksiyon herhangi bir değişkene atanabilir ve fonksiyonlar veri yapıları içerisinde tutulabilir. Bu yönüyle fonksiyonlar, JavaScript dilinin en önemli bileşenlerindendir. JavaScript dili hem fonksiyonel hem de nesne yönelimli bir dil özelliklerini barındırır. Ayrıca metin, dizi gibi veri yapılarıyla, düzenli ifadeler ve JSON verileri için uygulama programlama arayüzlerine sahiptir.

Dil adları, sözdizimi ve ilgili standart kütüphaneler de dahil olmak üzere JavaScript ve Java programlama dilleri arasında benzerlikler olsa dahi, bu iki programlama dili ayrı diller ve platformlardır.

2.5.2. JSON

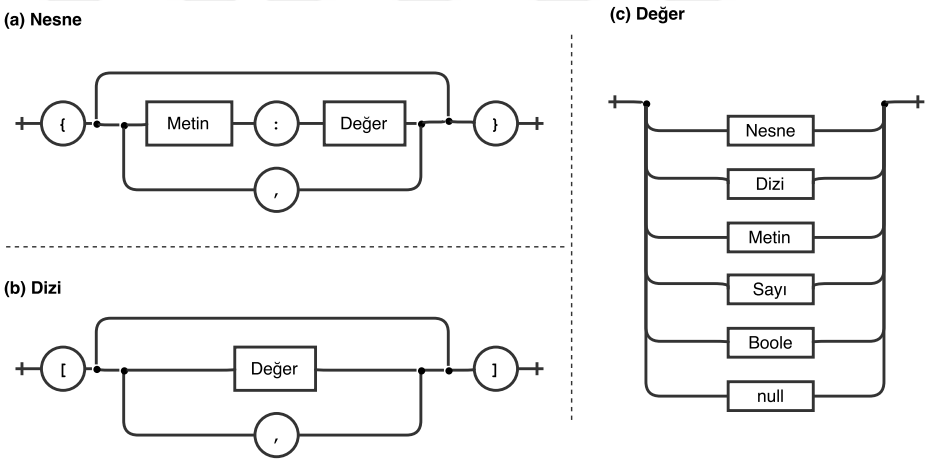
JSON (Javascript Object Notation) az yer kaplayan bir veri değişim formatıdır. [48] Sadece geliştiricilerin değil, sıradan insanların da rahatlıkla okuyup yorumlayabileceği kadar açık ve anlaşılır bir yazım formatına sahiptir. Ayrıca makinaların bu formatta saklanan verileri kolayca tarayıp kendi veri yapılarına dönüştürmesi çok kolaydır. JSON, programlama dillerinden bağımsızdır ancak yazımı JavaScript programlama dili veri yapılarının sözdizimiyle aynıdır. JSON formatının en önemli özelliği, farklı platformlar arasında veri biçimlerini taşımayı ve kullanmayı kolaylaştırmasıdır. Sözlük, dizi, metin gibi veri yapıları JSON formatına çevrilerek platformlar arasında kolaylıkla aktarılabilir. Çünkü JSON formatındaki verileri ayrıştırabilen kütüphaneler, neredeyse bütün programlama dillerinde ve bütün platformlarda yerleşik olarak bulunmaktadır.

JSON veri değişim biçimi iki temel yapı üzerine kurgulanmıştır:

- Bunlardan ilki, isim/değer çifti koleksiyonudur. Çeşitli programlama dillerinde bu yapı "nesne, yapı, sözlük" olarak tanımlanmıştır.
- İkincisi sıralı değer listesidir. Çoğu programlama dilinde bu yapı, "dizi,vektör, liste" olarak tanımlanmıştır.

Bu yapılar, genel standart veri yapılarıdır. Modern programlama dillerinin tamamı, bu yapıları, bir şekilde içlerinde barındırmaktadır. Programlama dilleri arasında veri değişimi için kullanılacak bir formatın, bu yapıları kullanarak oluşturulması kullanışlılık açısından gereklidir.

Bazı temel JSON veri türleri Şekil 2.10'da gösterilmiştir.

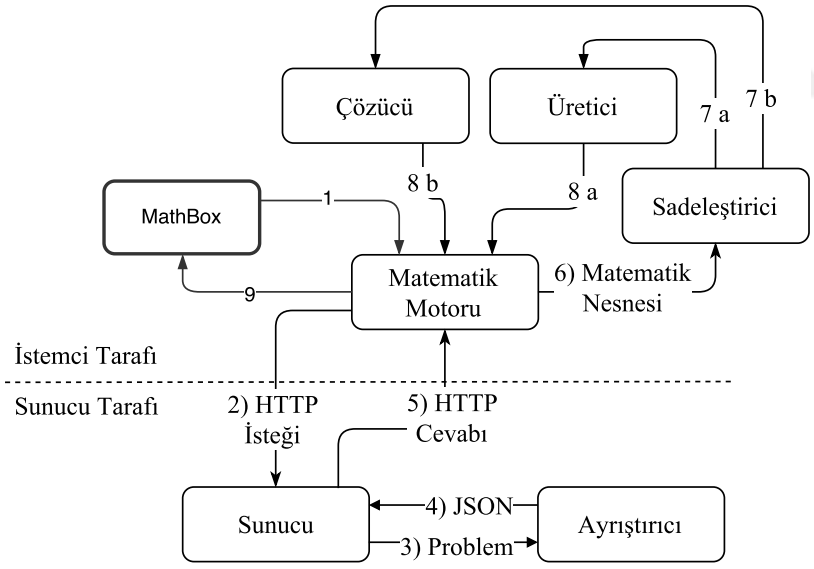


Şekil 2.10. Temel JSON veri türleri

3. UYGULAMA ÇATISININ GENEL YAPISI

Çalışma kapsamında geliştirilen uygulama çatısı altı temel bileşenden oluşmaktadır. Bunlar matematik motoru, sunucu, ayrıştırıcı, sadeleştirici, üretici ve çözücüdür. Matematik motoru tüm sistemin işleyişini yönetir. Sunucu matematik motorundan gelen polinom problemlerini ayrıştırıcıya iletir, bu ifadelerin JSON verilerine dönüştürülmüş halini ayrıştırıcıdan alır ve matematik motoruna geri döndürür. Sadeleştirici matematik motorundan gelen matematik nesnelerini en sade şekline getirir. Üretici sadeleştiriciden aldığı sade şekildeki probleme benzer yeni problemler üretir ve matematik motoruna iletir. Çözücü de aynı şekilde sade haldeki problemleri sadeleştiriciden alır, çözer ve matematik motoruna iletir. Matematik motoru bu işlemleri, çalışma kapsamında çatı üzerinde geliştirilen MathBox isimli uygulamanın isteği ile gerçekleştirir.

Uygulama çatısı ve MathBox için çalışma süreci Şekil 3.1'de gösterilmiştir.

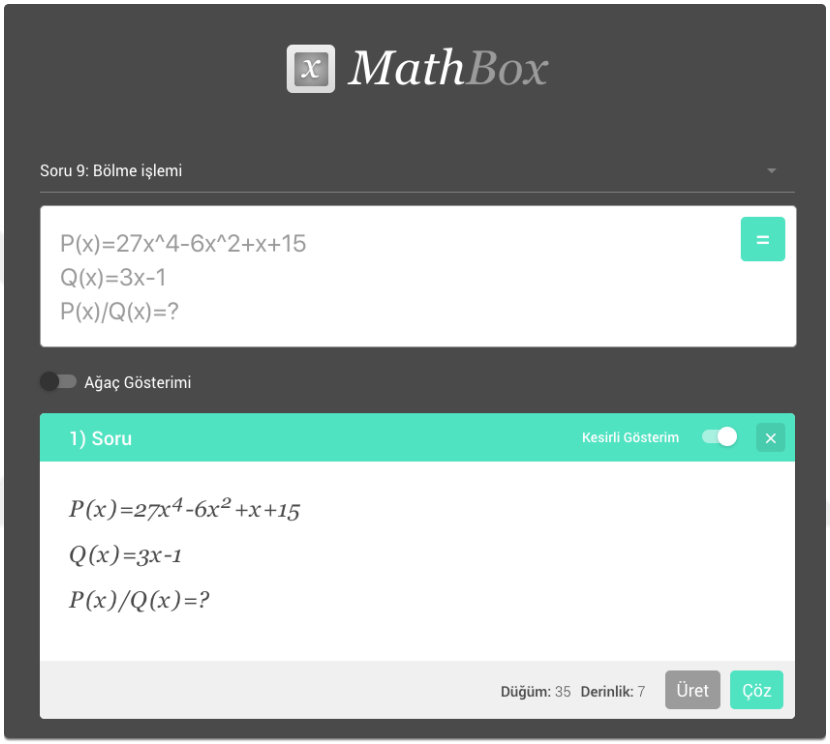


Şekil 3.1. Uygulama çatısı ve MathBox çalışma süreci

3.1. MathBox

Tez kapsamında geliştirilen uygulama çatısı kullanılarak, MathBox isimli bir web uygulaması geliştirilmiştir. Geliştirilen uygulamadan alınmış ekran görüntüleri Şekil 3.2, Şekil 3.3, Şekil 3.4 ve Şekil 3.5'de yer almaktadır.

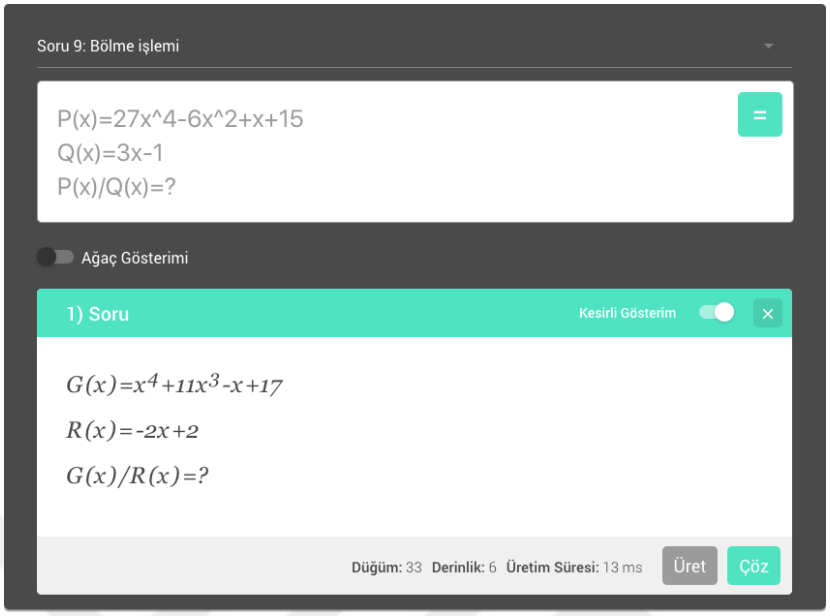
MathBox uygulamasının genel görünüşü, seçim kutusu yardımıyla bir problem türünün seçilmesi ve o türden bir problemin arayüzden alınması Şekil 3.2'de gösterilmiştir.



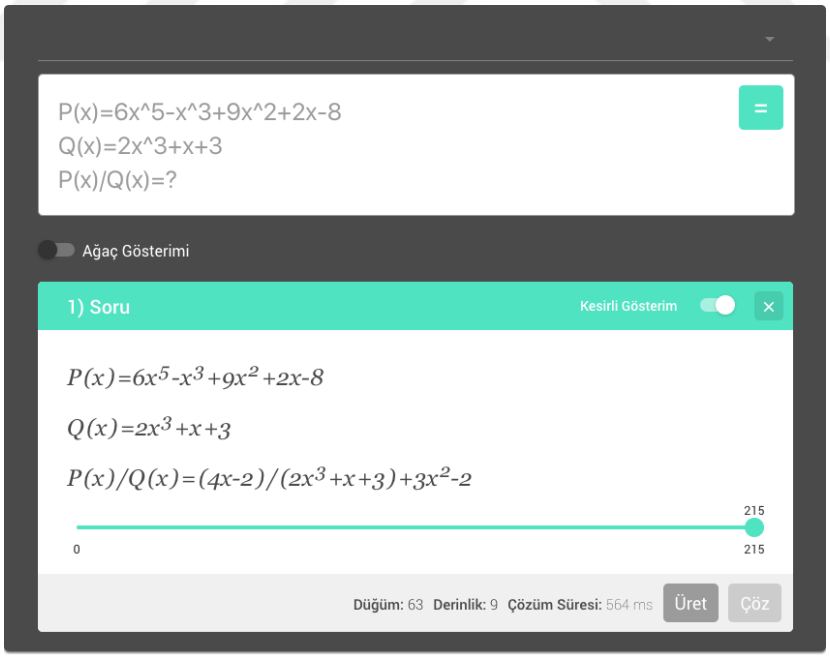
Şekil 3.2. Uygulamanın genel görünüşü

Şekil 3.2'den de görüldüğü gibi uygulama, Tablo 2.1'de gösterilen polinom problem türlerini adım adım çözebilir ve benzerlerini üretebilir. Polinom problemlerinde yer alan farklı sayı kümelerine ait (doğal sayı, rasyonel sayı gibi) sayılar üzerinde işlem yapabilir. Rasyonel sayıların, isteğe bağlı olarak kesirli veya ondalık sayı biçiminde gösterilmesini sağlayabilir. Bu görüntü, ağaç gösterimi kapalıyken alınmış bir görüntüdür. Ağaç gösterimi açıldığında, derinliği yedi, düğüm sayısı otuzbeş olan bir ağacın gösterileceği, en alt satırdaki bilgilerden anlaşılmaktadır.

MathBox uygulamasında, arayüzden alınan bir problemin benzerinin üretilmesi Şekil 3.3'de, adım adım çözülmesi Şekil 3.4'de gösterilmiştir.



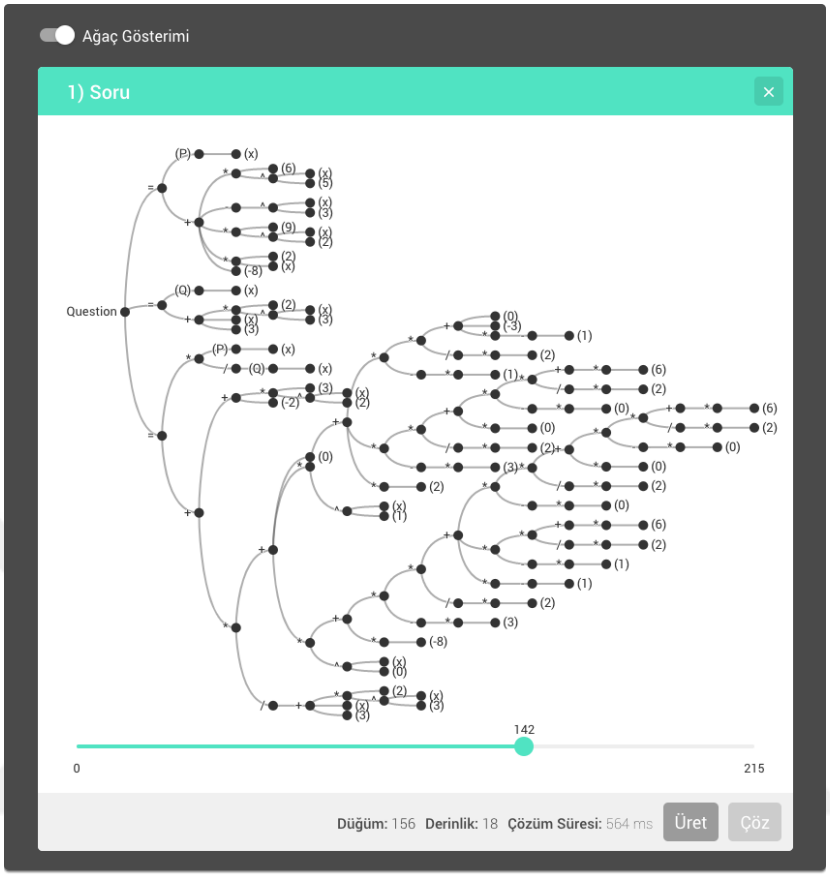
Şekil 3.3. Benzer problem üretimi



Şekil 3.4. Adım adım problem çözümü

Uygulama, polinom problemleri üzerinde yapılan benzer soru üretimi ve çözüm işlemleri sırasında, maksimum düğüm sayısı, derinlik ve süre bilgilerini ölçerek kullanıcıya gösterebilir.

MathBox uygulamasında, arayüzden okunan bir problemin adım adım çözülmesi sırasında, rastgele bir adımdan alınmış ağaç temsili görüntüsü Şekil 3.5'de gösterilmiştir.



Şekil 3.5. Rastgele seçilmiş bir adımın ağaç temsili

Uygulama, polinom problemlerinin çözümü veya benzerinin üretimi sırasında oluşan ağacın adım adım izlenmesine imkan verir. Uygulamanın animasyon yeteneği sayesinde, çözüm veya üretim süreci boyunca oluşan adımların takibini yapmak çok kolaydır. Burada ağaç gösterimi açık olduğu için, problem çözümünün her bir adımında oluşan ağaç rahatlıkla takip edilebilir.

4. POLİNOM PROBLEMLERİNİN ANALİZİ

İfade ayrıştırma, cebirsel ifadelerin alınmasıyla başlayıp sadeleştirilmesine kadar süren dört işlemden oluşur. Sistemin temel bileşenlerinden biri olan ayrıştırıcı, sunucudan gelen cebir problemlerini üç adımda JSON verisine dönüştürür. Diğer bileşen olan matematik motoru, ayrıştırıcıdan gelen JSON verisini matematik nesnelerine dönüştürür. Bu şekilde ayrıştırma işlemi tamamlanmış olur.

Geliştirilen arayüz yardımıyla kullanıcıdan alınan örnek bir problem 4.1'de gösterilmiştir.

$$\begin{aligned} P(x) &= x^2 + 3x - 7 \\ 5[P(x)^2] &= ? \end{aligned} \tag{4.1}$$

4.1. Sözcüksel Analiz

Belirli bir biçime sahip ifadeler için geliştiriciler tarafından önceden tanımlanmış veya kullanıcılar tarafından belirtilmiş özel anlamları bulunan sözcükler vardır. Bu sözcüklere token denir. Sözcüksel analiz bir arayüzden alınan kaynak verinin tokenlarına ayrıştırılmasıdır. Analiz aşamasında hata oluşmazsa alınan veri bir token dizisine dönüştürülür. [46]

Yapılan çalışmaya ait sözcüksel analiz algoritması aşağıdaki gibidir:

1. Ayrıştırıcı tarayıcıya token isteği yapar.
2. Tarayıcı kaynak veriden bir karakter okur ve bu karakterin gramer dosyasında tanımlı tokenlardan biriyle eşleşip eşleşmediğini kontrol eder.
3. Okunan karakterle başlayan tanımlı hiç bir token yoksa sözcüksel analizci sözcüksel hata verir ve analizi sonlandırır.
4. Okunan karakter token eşlemede yeterli olmadıysa her seferinde kaynak veriden bir karakter daha okuyarak token eşlemeye çalışır. Eşlediğinde tokenı ayrıştırıcıya yollar.

5. Eğer okuduğu yeni karakterle oluşan karakter dizisi için bir token eşleyemezse sözcüksel hata verir ve analizi sonlandırır.

6. Kaynak verinin okunması tamamlandığında en son elde edilecek olan ;end karakter dizisini alana kadar 1-5 arası adımlar tekrarlanır. ;end karakter dizisini alan ayrıştırıcı, tarayıcıya token isteği yapmayı sonlandırır.

Yapılan çalışmada yer alan token tanımlamalarının bazıları Tablo 4.1'de gösterilmiştir.

Tablo 4.1. Bazı token tanımlamaları

```
1  TOKEN :
2  {
3        <FNAME: ("P"|"Q"|"R"|"f"|"g"|"h")>
4     | <DEG: "der">
5     | <SQRT: "sqrt">
6     | <INT: "0"|(["1"-"9"])(["0"-"9"])*>
7     | <VAR: ("x"|"a"|"b"|"c"|"d"|"e")>
8     | <SIGN: "+"|"-" >
9     ...
10    | <QMARK: "?">
11    | <COM: ",">
12 }
```

$P(x) = x^2+3x-7 \quad 5[P(x)^2] =?$ için üretilen token dizisi Tablo 4.2'de gösterilmiştir.

Tablo 4.2. Örnek problem için üretilen token dizisi

İfade	**Token Dizisi**
P(x)=x^2+3x-7	(PNAME,"P") (LPR,"(")(VAR,"x")(RPR,")")(EQ,"=") (VAR,"x") (POW,"^")(INT,"2")(PLUS,"+")(INT,"3")(VAR,"x") (MINUS,"-")(INT,"7")
5[P(x)^2]=?	(INT,"5")(LSPR,"[")(PNAME,"P")(LPR,"(")(VAR,"x")(RPR,")") (POW,"^")(INT,"2")(RSPR,"]")(EQ,"=")(QMARK,"?")

4.2. Sözdizimsel Analiz

Çalışmada benzerlerinin üretilebileceği veya adım adım çözümlerinin yapılabileceği polinom problemleri analiz edilerek, JavaCC aracının derleyicisini üretebileceği bir bağlamdan bağımsız gramer Tablo 4.3'de gösterilen şekliyle önerilmiştir.

Tablo 4.3. Bağlamdan bağımsız gramer

$P \rightarrow$	$I+$
$I \rightarrow$	$E\text{“}=\text{”}(E\|\text{“}?\text{”})$
	$\|\text{“}der\text{”}\ \text{“}[\text{”}\ E\text{“}]\text{”}\ \text{“}=\text{”}(N\|\text{“}?\text{”})$
$E \rightarrow$	$T((\text{“}+\text{”}\|\text{“}-\text{”})T)*$
$T \rightarrow$	$U(\text{“}*\text{”}T\|\text{“}/\text{”}U)*$
$U \rightarrow$	$((\text{“}+\text{”}\|\text{“}-\text{”})Ti)$
	$\|Ti$
$Ti \rightarrow$	$Po(Po)*$
$Po \rightarrow$	$El(\text{“}\hat{}\text{”}\ Po)?$
$El \rightarrow$	N
	$\|V$
	$\|\text{“}(\text{”}\ E\text{“})\text{”}$
	$\|\text{“}sqrt\text{”}\ \text{“}(\text{”}\ E\text{“})\text{”}$
	$\|\text{“}nrt\text{”}\ \text{“}(\text{”}\ E\text{“},\text{”}\ E\text{“})\text{”}$
	$\|(\text{“}P\text{”}\|\text{“}Q\text{”}\|\text{“}R\text{”}\|\text{“}f\text{”}\|\text{“}g\text{”}\|\text{“}h\text{”})\ \text{“}(\text{”}\ E\text{“})\text{”}$
$N \rightarrow$	$\text{“}0\text{”}\|(([\text{“}1\text{”}\ -\ \text{“}9\text{”}])([\text{“}0\text{”}\ -\ \text{“}9\text{”}])*)$
$V \rightarrow$	$\text{“}x\text{”}\|[\text{“}a\text{”}\ -\ \text{“}e\text{”}]$
$Q \rightarrow$	$\text{“}?\text{”}$

Burada P : parse, I : input, E : exp, T : term, U : unary, Ti : times, Po : power, El : element, V : var ve Q : qmark'ı temsil etmektedir.

Ayrıştırıcı geliştirilirken sözcüksel analizciden gelen token dizisinin hangi öncelik sırasıyla ve hangi yönden başlanarak işleme sokulacağı önceden belirlenmelidir. Bu kuralların, sözdizimi kuralları, belirlendiği yer sözdizimsel analiz kısmıdır. Sözdizimsel analiz aşamasında kelimesel analiz kısmından gelen token dizisinin önceden tanımlanan kurallara uygunluğu kontrol edilir. Analiz aşamasında hata oluşmazsa gelen token dizisi tanımlanan kurallara uygun bir soyut sözdizim ağacına dönüştürülür.

Yapılan çalışmaya ait sözdizimsel analiz algoritması aşağıdaki gibidir:

1. Ayrıştırıcı tarayıcıdan belirli sayıda token ister.
2. Tarayıcıdan gelen token dizisinden ilk tokendan başlayarak her seferinde bir token okur.
3. Okunan tokenla başlayan tanımlı hiç bir sözdizim kuralı yoksa sözdizimsel hata verir ve analizi sonlandırır.
4. Okunan ilk token, bir kurala eşleme yapmada yeterli olmadıysa, her seferinde token dizisinden bir token daha okuyarak önceden tanımlanan sözdizim kurallarına eşleme yapmaya çalışır. Eşleme yaptığı anda ilgili sözdizim kuralına uygun nesne ağacını oluşturur.
5. Ayrıştırıcı tüm token dizisini bir soyut sözdizim ağacına dönüştürene kadar 2-4 arası adımları tekrarlar.

Soyut sözdizim ağacındaki her düğüm bir sözdizim sınıfıyla temsil edilmektedir. Çalışmada yer alan bazı sözdizim sınıfı örnekleri Tablo 4.4'de gösterilmiştir.

Tablo 4.4. Bazı sözdizim sınıfı örnekleri

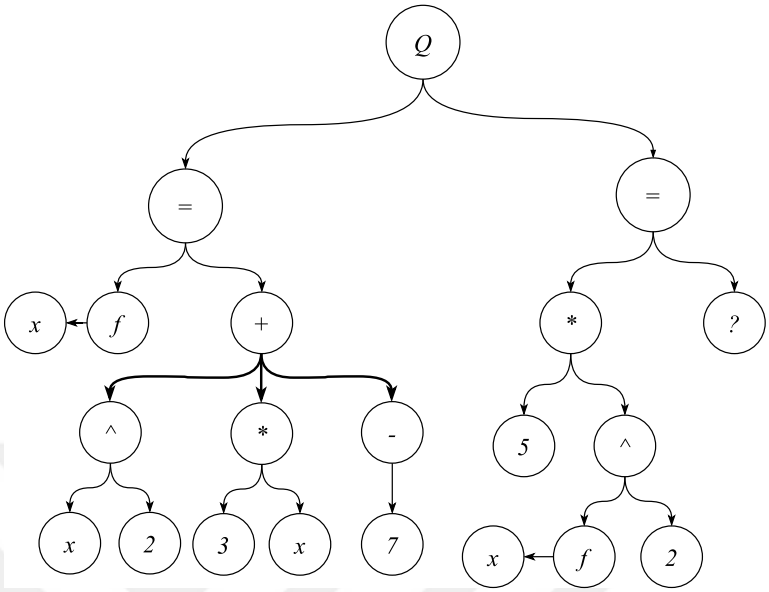
```
1  class Exp() { }
2  ...
3  class Times extends Exp {
4  public List<Exp> terms;
5  public Times(Exp ... exps) {
6  for(Exp exp: exps){ terms.add(exp); }}...
7  }
8  ...
9  class Var extends Exp {
10      public String var;
11      public Var(String _name){ name = _name; }...
12 }
```

$P(x) = x^2+3x-7 \quad 5[P(x)^2] =?$ probleminin nesne ağacı Tablo 4.5'de gösterilmiştir.

Tablo 4.5. Örnek problem için üretilen nesne ağacı

```
1  Question question = new Question(
2  new Equal(new Polynomial( "P", new Var("x")),
3   new Plus(new Pow(new Var("x"),new Const(2)),
4    new Times(new Const(3),new Var("x")),
5    new Minus(new Const(7)) )),
6  new Equal(new Times(new Const(5),new Pow(
7     new Polynomial("P",new Var("x")),new Const(2)) ),
8   new QMark() )
9  )
```

Tablo 4.5'de gösterilen nesne ağacının soyut sözdizim ağacı şeklinde temsili Şekil 4.1'de gösterilmiştir.



Şekil 4.1. Örnek problem için üretilen soyut sözdizim ağacı

4.3. Soyut Sözdizim Ağacının JSON Verisine Dönüştürülmesi

Geliştirilen ayrıştırıcı metin formatında alınan matematiksel ifadeleri soyut sözdizim ağacına dönüştürür. Ayrıştırıcılar belirli bir platforma yönelik ara kod ürettiklerinden, soyut sözdizim ağaçlarının değerlendirilmesi sadece ilgili platformda yapılabilir. JSON ise dizi, sözlük gibi birçok farklı veri yapısının farklı yazılım platformları arasında değiş tokuşuna olanak sağlayan bir formattır. Bu nedenle platformlar arası değerlendirme süreçlerine destek verebilmek için ayrıştırıcı son olarak elde ettiği sözdizim ağacını JSON formatındaki veriye dönüştürür ve sadeleştiriciye yollaması için sunucuya iletir.

Soyut sözdizim ağacını JSON verisine dönüştürme algoritması aşağıdaki gibidir:

1. Ayrıştırıcıdan gelen soyut sözdizim ağacının kök düğümü alınır.
2. Düğümün üye olduğu sözdizim sınıfını temsil eden JSON verisi oluşturulur.
3. Düğümün çocuk düğümü varsa; (ilk çocuk düğüm için i= 1 olmak üzere) i. çocuk düğüm için 2. adıma geri dönülür.

4. Düğümün çocuk düğümü yoksa 2. adımda düğüm için elde edilen JSON verisi düğümün ebeveynine döndürülür.

5. Ebeveyn düğüm tüm çocuk düğümlerinden JSON verilerini alana kadar i'yi her seferinde bir arttırarak 3. adımı çalıştırır. Tüm çocuk düğümlerinin JSON verisini aldığında kendisi için 2. aşamada oluşturulan JSON verisine bu verileri ekleyerek kendi ebeveynine yollar ve ebeveyni de tüm çocuk düğümlerini dolaşmak için 3. adımda kaldığı yerden (i'yi bir attırarak) devam eder.

6. İncelenen düğümün ebeveyni yoksa (kök düğümse) dönüştürme süreci tamamlanmış olur.

Tablo 4.5'de gösterilen ağacın JSON verisi hali Tablo 4.6'da gösterilmiştir.

Tablo 4.6. Örnek problem için üretilen JSON verisi

```
{
"type": "question",
"items": [
 {  "type": "equal", "items": [...] },
 {
  "type": "equal",
  "items": [
   {
    "type": "times",
    "items": [
     { "type": "const", "value": 5 },
     { "type": "power",
   "items": [
     { "type": "polynomial", "name": "P", "items": [{ "type":  "
         var", "name": "x" }] },
     { "type": "const", "value": 2}]    }]   },
   { "type": "qmark" } ] }]
}
```

4.4. Matematik Nesneleri

Sunucudan HTTP cevabı aracılığıyla alınan JSON verileri, sadeleştirme, üretim ve çözüm işlemlerine destek vermek için matematik nesnelerine dönüştürülür. Matematik nesnelerinin bu işlemleri kolaylaştırması, sözdizim ağaçlarındaki herbir sınıfa dönüşüm sırasında eklenen benzersiz tanımlayıcılar sayesinde olur. Ayrıca bu tanımlayıcılar, sınıflar arasındaki ilişkileri tanımlarken de kolaylık sağlar.

Örnek matematik nesnesi sınıf yapısı Tablo 4.7'de gösterildiği gibidir.

Tablo 4.7. Sözde kod : örnek matematik nesnesinin sınıf yapısı

```
1  Sinif MatematikNesnesi {
2   id: Metin
3   cocukDugumler: [matematikNesnesi]
4
5   uret()
6   sadelestir()
7   ortakParantezeAl()
8   dugumuBul()
9   ...
10 }
```

5. POLİNOM PROBLEMLERİNİN SADELEŞTİRİLMESİ

Sadeleştirme işlemleri, ifadeleri birbiriyle karşılaştırabilme, benzerliklerini tespit etme gibi birçok işlemde kolaylık sağlar. Bu yüzden sadeleştirme işlemlerinin tüm sınıf yapıları içerisinde yapılması gerekmektedir. Yapılan her işlem için sadeleştirme işleminin tekrarlanması sonucunda, ifade en sade şekline getirilmiş olur.

Çalışmada yer alan sözdizim sınıflarının herbirinin içerisinde, kendisiyle ilgili sadeleştirme metodu bulunmaktadır. Bu sebeple, bir matematik nesnesi için sadeleştirme süreci başlatıldığında, her sınıf kendisiyle ilgili düğümle ilgilenir. Tablo 5.1'de de gösterilen sadeleştirme süreci, bir matematik nesnesi en sade hale gelince sonlanacak şekilde geliştirilmiştir. Bir nesne sadeleştirilirken her yinelemeden önce ve sonra ilgili nesnenin birer kopyası oluşturulur. Her yinelemeden sonra bu kopyalar karşılaştırılır. Kopyalar tamamen birbirinin aynısı çıktığında yineleme süreci durdurulur ve sadeleştirme sonucu elde edilmiş olur.

Tablo 5.1. Özyinelemeli sadeleştirme süreci

```
1  soru = Soru(JSONVerisi)
2
3  sonsuz dongu:
4   once <- soru.metneCevir();
5   soru.sadelestir()
6   sonra <- soru.metneCevir();
7   eger once esitse sonra
8    dur
```

5.1. Temel Sınıf Yapıları İçerisinde Yapılan Sadeleştirmeler

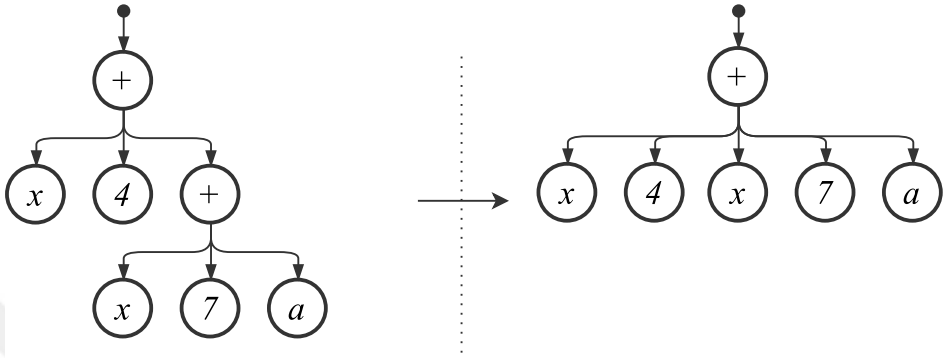
5.1.1. Toplama Sınıfı

Tablo 5.2. Sözde kod : toplama sınıfına ait nesneleri sadeleştirme

```
1  Sinif Toplama <- matematikNesnesi {
2   sadelestir() {
3    ustsinif.sadelestir()
4    her cocukDugum <- cocukDugumler:
5     eger cocukDugum Toplama sinifina uyeyse:
6      cocukDugum.cocukDugumler ->> cocukDugumler
7    `cocukDugumler`i sirala
8    yeniCocukDugumler = []
9    her cocukDugum <- cocukDugumler:
10    eger yeniCocukDugumler bossa:
11     cocukDugum ->> yeniCocukDugumler
12    degilse:
13     her yeniCocukDugum <- yeniCocukDugumler:
14      eger yeniCocukDugum ve cocukDugum Sayi sinifina uyeyse:
15       yeniCocukDugum.deger = cocukDugum.deger + yeniCocukDugum.
             deger
16      eger yeniCocukDugum.metneCevir() == cocukDugum.metneCevir():
17       yeniCocukDugum degistir Carp(cocukDugum, Sayi(2))
18      degilse:
19       cocukDugum ->> yeniCocukDugumler
20   cocukDugumler <- yeniCocukdugumler
21   eger sadece bir cocuk dugum varsa:
22    dondur cocukDugumler tek elemanini
23   degilse
24    0 degerli nesneler cocukDugumler listesinden cikarilir
25    dondur kendisini
26  }
27 }
```

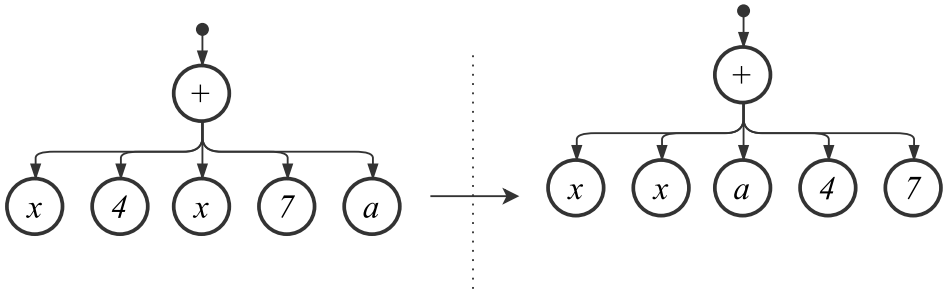
Toplama sınıfına ait nesneleri sadeleştirebilmek için geliştirilen programın sözde kodu Tablo 5.2'de gösterilmiştir.

Tablo 5.2'de sözde kodu gösterilen program yardımıyla yapılan sadeleştirme işlemi örnekleri Şekil 5.1, Şekil 5.2, Şekil 5.3, Şekil 5.4 ve Şekil 5.5'de gösterilmiştir.

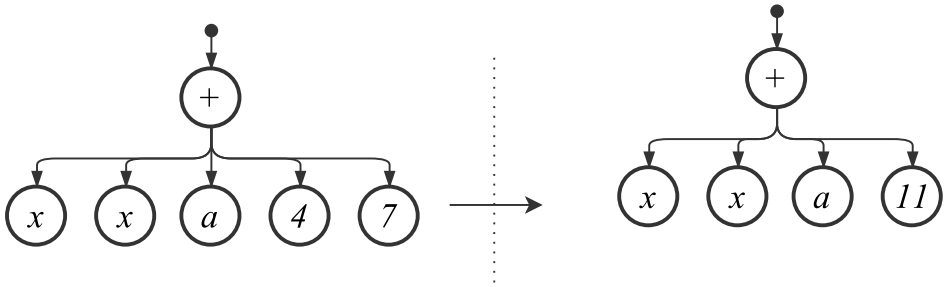


Şekil 5.1. $x + 4 + (x + 7 + a) \rightarrow x + 4 + x + 7 + a$

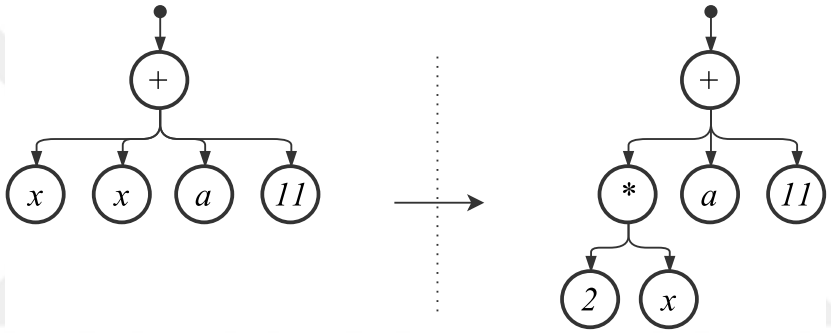
Tablo 5.2'de bahsedilen sıralama işlemine göre, toplama sınıfı nesnelerinin elemanları üs, çarpma, değişken ve sayı sınıfı sırasına uygun olacak şekilde sıralanır.



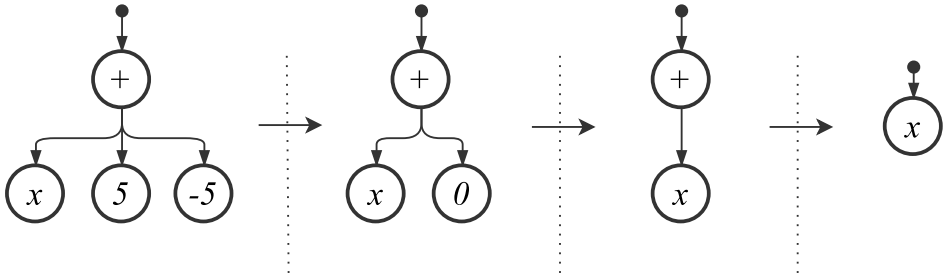
Şekil 5.2. $x + 4 + x + 7 + a \rightarrow x + x + a + 4 + 7$



Şekil 5.3. $x + x + a + 4 + 7 \rightarrow x + x + a + 11$



Şekil 5.4. $x + x + a + 11 \rightarrow 2 * x + a + 11$



Şekil 5.5. $x + 5 - 5 \rightarrow x$

5.1.2. Çarpma Sınıfı

Çarpma sınıfına ait nesneleri sadeleştirebilmek için geliştirilen programın sözde kodu Tablo 5.3 ve Tablo 5.4'de gösterilmiştir.

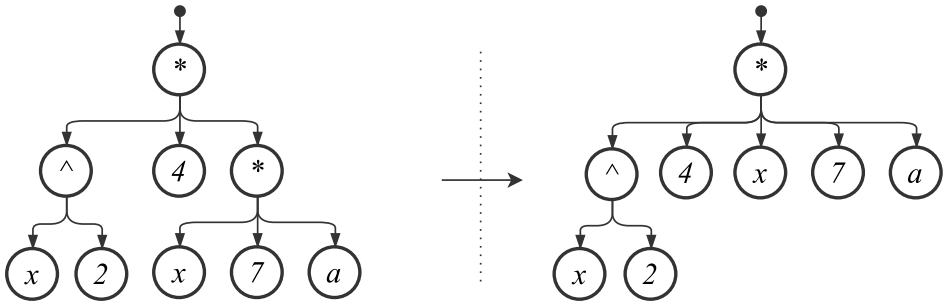
Tablo 5.3. Sözde kod : çarpma sınıfına ait nesneleri sadeleştirme

```
1  Sinif Carpma <- matematikNesnesi {
2   sadelestir() {
3    ustsinif.sadelestir()
4    her cocukDugum <- cocukDugumler:
5     eger cocukDugum Carpma sinifina uyeyse:
6      cocukDugum.cocukDugumler ->> cocukDugumler
7    `cocukDugumler`i sirala
8    yeniCocukDugumler = []
9    her cocukDugum <- cocukDugumler:
10    eger yeniCocukDugumler bossa:
11     cocukDugum ->> yeniCocukDugumler
12    degilse:
13     her yeniCocukDugum <- yeniCocukDugumler:
14      eger yeniCocukDugum ve cocukDugum Sayi sinifina uyeyse:
15       yeniCocukDugum.deger = cocukDugum.deger * yeniCocukDugum.
             deger
16      eger yeniCocukDugum ve cocukDugum Degisken sinifina uyeyse ve
             ikisinin ismi ayniysa:
17       yeniCocukDugum degistir Us(cocukDugum, Sayi(2))
18      eger yeniCocukDugum ve cocukDugum Us sinifina uyeyse ve
            tabanlari esitse:
19       yeniCocukDugum degistir Us(cocukDugum.taban(),
             yeniCocukDugum.us + cocukDugum.us)
20
21    ...
```

Tablo 5.4. Sözde kod (devamı) : çarpma sınıfına ait nesneleri sadeleştirme

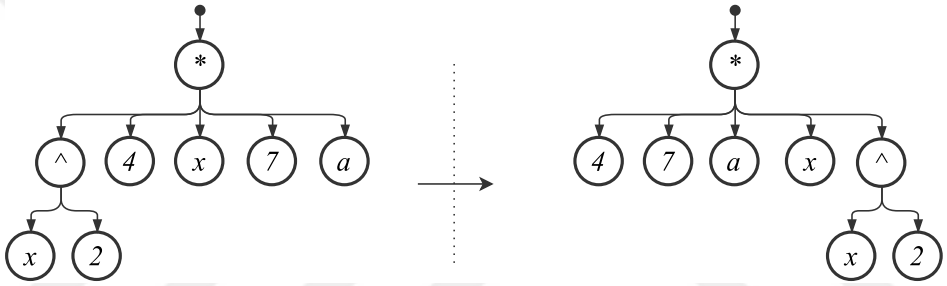
```
1     ...
2         eger cocukDugum Bolme sinifina uyeyse ve cocukDugum.
              cocukDugumler[0].metneCevir() == yeniCocukDugum.
              metneCevir():
3          yeniCocukDugum degistir Sayi(1)
4         eger cocukDugum Toplama sinifina uyeyse:
5          yeniCocukDugum degistir (cocukDugum ile yeniCocukDugum
               kartezyen carpilir)
6         degilse:
7          cocukDugum –>> yeniCocukDugumler
8     cocukDugumler <– yeniCocukdugumler
9     eger sadece bir cocuk dugum varsa:
10     dondur cocukDugumler tek elemanini
11    eger ‘cocukDugumler‘den herhangi biri 0 ise:
12     dondur Sayi(0)
13    degilse
14     sayi sinifina ait 1 degerli nesneler cocukDugumler listesinden
          cikarilir
15     dondur kendisini
16   }
17 }
```

Tablo 5.3 ve Tablo 5.4'de sözde kodu gösterilen program yardımıyla yapılan sadeleştirme işlemi örnekleri Şekil 5.6, Şekil 5.7, Şekil 5.8, Şekil 5.9, Şekil 5.10, Şekil 5.11 ve Şekil 5.12'de gösterilmiştir.

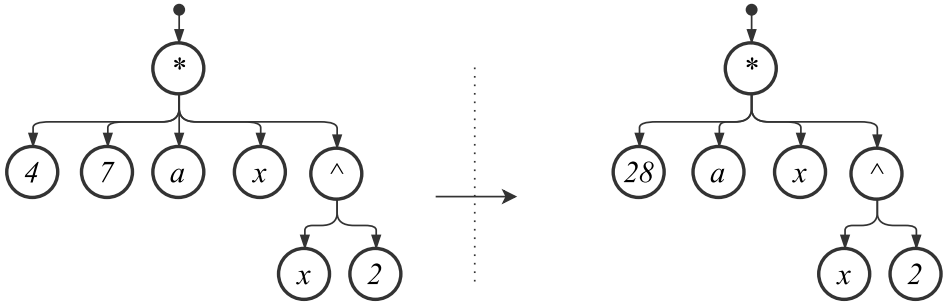


Şekil 5.6. $x^2 * 4 * (x * 7 * a) \rightarrow x^2 * 4 * x * 7 * a$

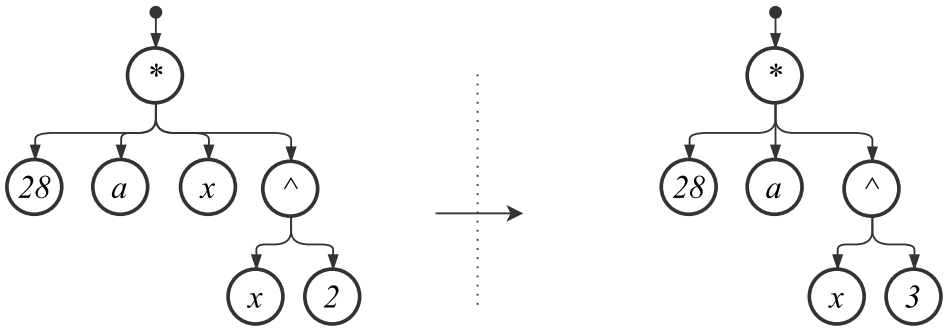
Tablo 5.3'de bahsedilen sıralama işlemine göre, çarpma sınıfı nesnelerinin elemanları sayı, değişken, üs, bölme sınıfı sırasına uygun olacak şekilde sıralanır.



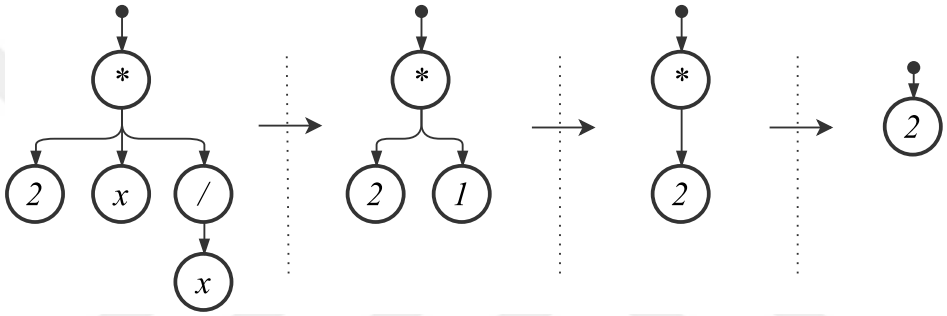
Şekil 5.7. $x^2 * 4 * x * 7 * a \rightarrow 4 * 7 * a * x * x^2$



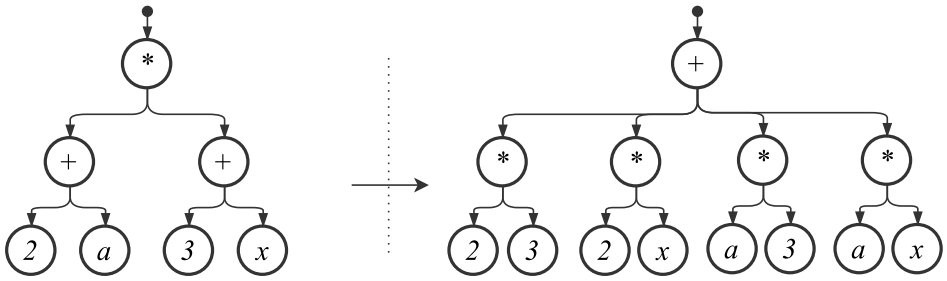
Şekil 5.8. $4 * 7 * a * x * x^2 \rightarrow 28 * a * x * x^2$



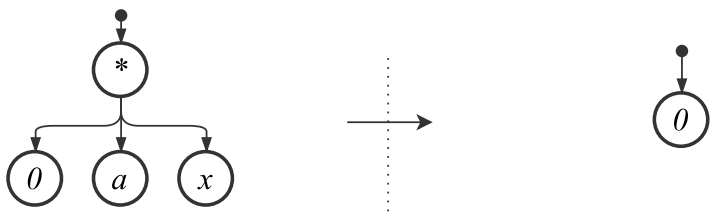
Şekil 5.9. $28 * a * x * x^2 \rightarrow 28 * a * x^3$



Şekil 5.10. $2 * x(/x) \rightarrow 2$



Şekil 5.11. $(2 + a) * (3 + x) \rightarrow (2 * 3) + (2 * x) + (a * 3) + (a * x)$



Şekil 5.12. $0 * a * x \rightarrow 0$

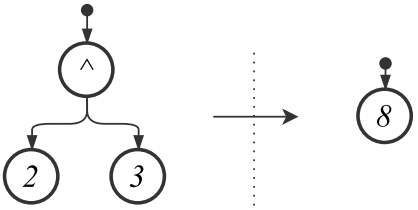
5.1.3. Üs Sınıfı

Üs sınıfına ait nesneleri sadeleştirebilmek için geliştirilen programın sözde kodu Tablo 5.5'de gösterilmiştir.

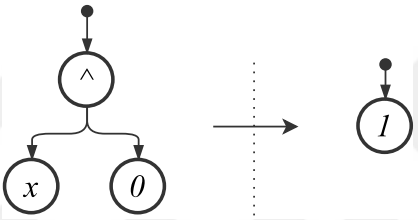
Tablo 5.5. Sözde kod : üs sınıfına ait nesneleri sadeleştirme

```
1  Sinif Us <- matematikNesnesi {
2   us: matematikNesnesi
3   taban: matematikNesnesi
4   sadelestir() {
5    ustsinif.sadelestir()
6    eger taban ve us Sayi sinifi uyesiyse:
7     dondur Sayi(taban.deger ^ us.deger)
8    eger us Sayi sinifina uyeyse ve degeri 0:
9     dondur Sayi(1)
10   eger us Sayi sinifina uyeyse ve degeri 1:
11    dondur taban
12   eger taban Carpma sinifina uyeyse:
13    dondur Carpma( her Us(cocukDugum, us) <- taban.cocukDugumler )
14   eger taban Us sinifina uyeyse:
15    dondur Us( taban.taban, Carpma(taban.us, us) )
16   eger us Toplama sinifina uyeyse:
17     dondur Us( her Us(taban, cocukDugum) <- us.cocukDugumler )
18   eger taban Cikarma sinifina uyeyse ve us Sayi sinifina uyeyse:
19    eger us.deger cift sayiysa:
20     dondur Us(taban.icDugum, us.deger)
21    degilse:
22      dondur Cikarma(Us(taban.icDugum, us.deger))
23   eger taban Toplama sinifina uyeyse ve us Sayi sinifina uyeyse:
24    dondur taban ve us ile binom acilimi yapilir
25   degilse: dondur kendisi
26  }
27 }
```

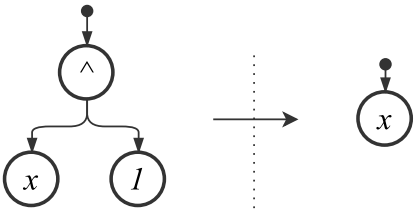
Tablo 5.5'de sözde kodu gösterilen program yardımıyla yapılan sadeleştirme işlemi örnekleri Şekil 5.13, Şekil 5.14, Şekil 5.15, Şekil 5.16, Şekil 5.17, Şekil 5.18, Şekil 5.19 ve Şekil 5.20'de gösterilmiştir.



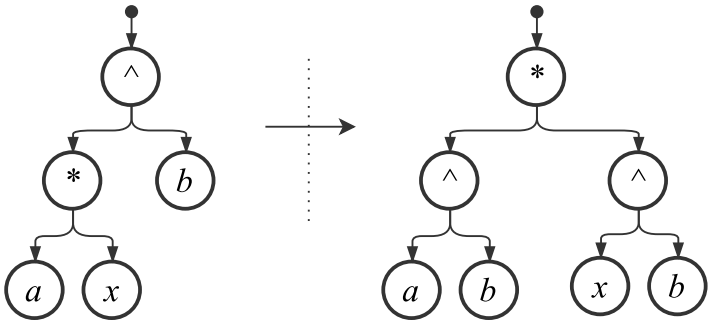
Şekil 5.13. $2^3 \rightarrow 8$



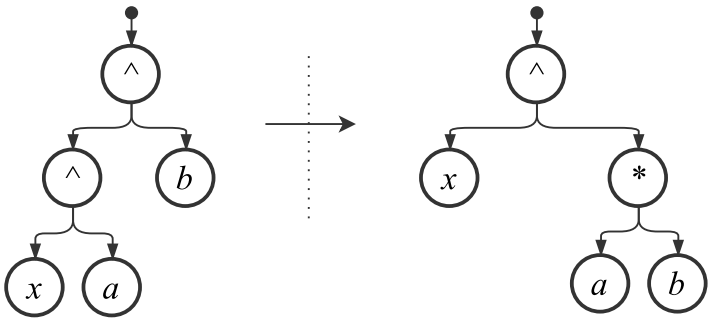
Şekil 5.14. $x^0 \rightarrow 1$



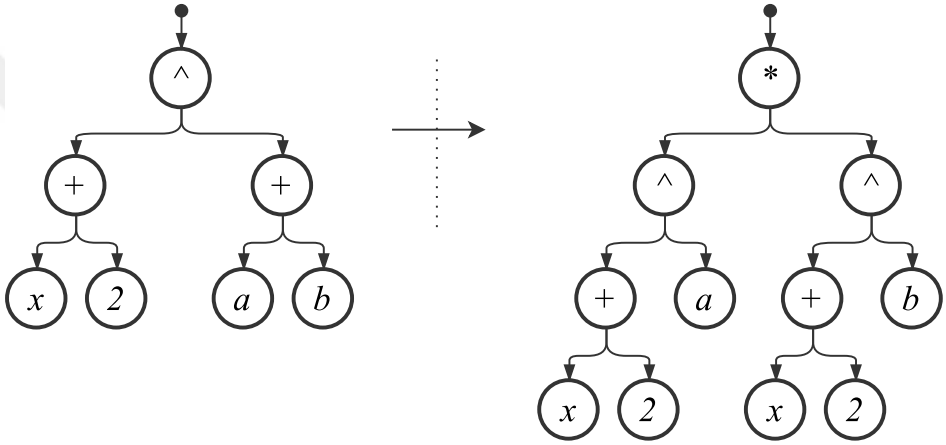
Şekil 5.15. $x^1 \rightarrow x$



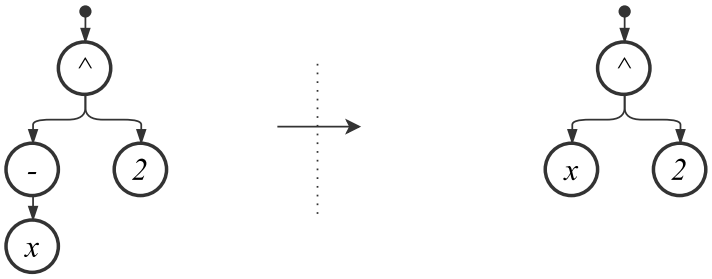
Şekil 5.16. $(a * x)^b \rightarrow a^b * x^b$



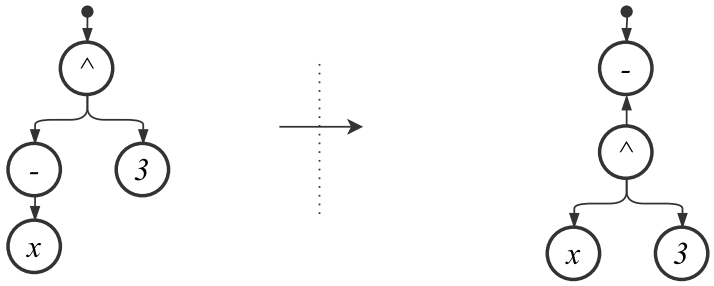
Şekil 5.17. $(x^a)^b \rightarrow x^{a*b}$



Şekil 5.18. $(x+2)^{a+b} \rightarrow (x+2)^a * (x+2)^b$



Şekil 5.19. $(-x)^2 \rightarrow x^2$



Şekil 5.20. $(-x)^3 \rightarrow -(x^3)$

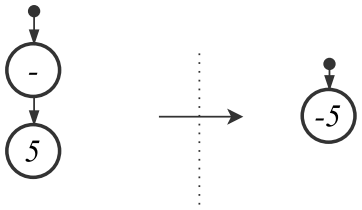
5.1.4. Çıkarma Sınıfı

Çıkarma sınıfına ait nesneleri sadeleştirebilmek için geliştirilen programın sözde kodu Tablo 5.6'de gösterilmiştir.

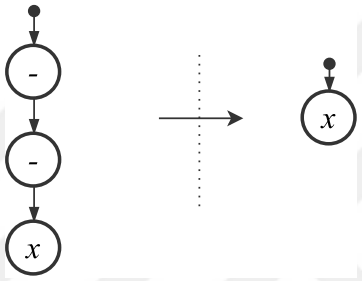
Tablo 5.6. Sözde kod : çıkarma sınıfına ait nesneleri sadeleştirme

```
1  Sinif Cikarma <- matematikNesnesi {
2   icDugum: matematikNesnesi
3   sadelestir() {
4    ustsinif.sadelestir()
5    eger icDugum Sayi sinifi uyesiyse:
6     dondur Sayi(-1 * icDugum.deger)
7    eger icDugum Cikarma sinifi uyesiyse:
8     dondur icDugum.icDugum
9    degilse:
10    dondur kendisi
11  }
12 }
```

Tablo 5.6'de sözde kodu gösterilen program yardımıyla yapılan sadeleştirme işlemi örnekleri Şekil 5.21 ve Şekil 5.22'de gösterilmiştir.



Şekil 5.21. $-(5) \rightarrow -5$



Şekil 5.22. $-(-(x)) \rightarrow x$

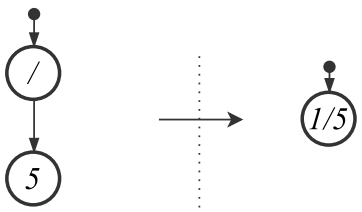
5.1.5. Bölme Sınıfı

Bölme sınıfına ait nesneleri sadeleştirebilmek için geliştirilen programın sözde kodu Tablo 5.7'de gösterilmiştir.

Tablo 5.7. Sözde kod : bölme sınıfına ait nesneleri sadeleştirme

```
1  Sinif Bolme <- matematikNesnesi {
2   icDeger: matematikNesnesi
3   sadelestir() {
4    ustsinif.sadelestir()
5    eger icDeger Sayi sinifina uyeyse:
6     dondur Sayi(1/icDeger.deger)
7    degilse:
8     dondur kendini
9   }
10 }
```

Tablo 5.7'de sözde kodu gösterilen program yardımıyla yapılan sadeleştirme işlemi örneği Şekil 5.23'de gösterilmiştir.



Şekil 5.23. $/(5) \rightarrow 1/5$

5.2. Özel Metodlar Yardımıyla Yapılan Sadeleştirmeler

5.2.1. Pascal Üçgeninin Hesaplanması

Tabanı toplama, üssü sayı sınıfına ait nesneler olan bir üs sınıfı nesnesi, doğrudan yapılacak bir sadeleştirme işlemine uygun değildir. Doğrudan sadeleştirilemeyen bu tür ifadeler, sadeleştirme süreci sonucunda elde edilen ifadenin en sade şekilde olmasını engellemektedir. Bu ifadelerin sadeleştirme sürecine dahil edilebilmesi için, öncelikle toplama sınıfı nesnelerine dönüştürülmeleri gerekir. Bu durum üs sınıfı nesnelerinin Binom açılımı uygulanarak toplama sınıfı nesnelerine dönüştürülmesiyle mümkün olur. Nesnelerin Binom açılımı yapılırken, toplama sınıfının elemanı olacak her bir çarpma sınıfı nesnesinin başına gelecek katsayılar, Pascal üçgeni yardımıyla bulunur.

5.2.2. Binom Açılımının Yapılması

Binom açılımı yapılacak üs sınıfı nesnesinin tabanındaki toplama nesnesinde ikiden fazla eleman olabilir. Böyle durumlarda binom açılımı yapabilmek için önce tabandaki toplama sınıfı nesnesinin elemanları, üs, çarpma, değişken ve sayı sınıfı sırasına uygun olacak şekilde sıralanır. Ardından her yinelemede sıralı haldeki elemanlardan ilki birinci terim, kalan tüm elemanlar ikinci terim kabul edilerek binom açılımı yapılır.

Örneğin, $(x + 2 + a)^3$ üs sınıfı nesnesinde taban $x + 2 + a$ ve üs 3'dür. Bu nesne için Pascal üçgeninden gelen katsayılar $1, 3, 3$ ve 1'dir. İlk olarak üs sınıfı nesnesinin tabanındaki toplama sınıfı nesnesinin eleman sayısı ikiden fazla olduğu için $(x+a+2)$ şeklinde sıralanır. Ardından birinci terim olarak x, ikinci terim olarak $(a + 2)$ alınır ve Binom açılımı yapılır. İlk yinelemede elde edilen sonuç aşağıdaki şekildedir:

$x^3 + 3x^2(a + 2) + 3x(a + 2)^2 + (a + 2)^3$

Elde edilen ifade, temel sınıf yapıları içerisinde tanımlanan sadeleştirme kurallarıyla sadeleştirilebilecek hale gelmemiştir. Bu yüzden yinelemeli Binom açılımı sürecine devam edilir. Binom açılımı süreci ve temel sınıf yapıları içerisinde tanımlanan sadeleştirme işlemleri tamamlandıktan sonra elde edilen sonuç aşağıdaki şekildedir:

$x^3 + (3a + 6)x^2 + (3a^2 + 12a + 12)x + a^3 + 6a^2 + 12a + 8$

5.2.3. Benzer İfadelerin Eliminasyonu

Çarpımların toplamı şeklindeki ifadelerin içindeki çarpımların, içerdiği en yüksek dereceli ortak çarpanın parantezine alınmasıyla, fazladan yapılacak çarpma işlemi sayısının azaltılacağı 1800'lü yılların başlarında öne sürülmüştür. [50] Bilgisayarların ortaya çıkışıyla bilgisayar ortamında problem çözme yaygınlaşmış, önceleri elle problem çözerken yapılan işlemler, bilgisayar ortamına da aktarılmaya başlanmıştır. Sadeleştirme işlemleri de bu işlemlerden biridir. Bilgisayar ortamına ilk aktarılan sadeleştirme işlemlerinden biri çalışmada da yer bulan en büyük ortak çarpanın parantezine alma işlemidir. Horner metodu olarak da bilinen bu işlem özellikle polinom problemlerinin çözüm hızını iyileştirmede kullanılır. Bu metod yardımıyla yapılan örnek bir sadeleştirme işlemi aşağıdaki gibidir:

$ax^3 + 5x^2 + 7bx^3 \rightarrow x^2(ax + 5 + 7bx)$

Bilgisayar ortamına ilk aktarılan sadeleştirme işlemlerinden bir diğeri, benzer toplama ve çarpma ifadelerinin parantezlenmesi (global common subexpression elimination) işlemidir. [39] Bu işlem sayesinde yapılması gereken toplama ve çarpma işlemlerinin sayısı azaltılmış olur.

6. POLİNOM PROBLEMLERİNİN ÜRETİLMESİ

Tüm problemlerin içerisinde değişime uygun olan ve olmayan kısımlar vardır. Kullanıcılardan bir arayüz yardımıyla alınan polinom problemlerinin benzerleri üretilirken, problemlerin temel yapıları korunmalıdır.

Üretim işlemine uygun olan sözdizim sınıflarının her birinin içerisinde, kendi sınıfıyla ilgili benzer ifade üretmeyi sağlayan üretim metodu bulunmaktadır. Bu sebeple, bir matematik ifadesi için benzer ifade üretme süreci başlatıldığında, her sözdizim sınıfı kendi sınıfıyla ilgili düğümlerle ilgilenir.

6.1. Soru Sınıfı

Soru sınıfına ait nesnelerin benzerlerini üretebilmek için geliştirilen programın sözde kodu Tablo 6.1'de gösterilmiştir.

Tablo 6.1. Sözde kod : soru sınıfına ait nesnelerin benzerlerini üretme

```
1  Sinif Soru <- MatematikNesnesi {
2   uret() {
3    yenisoru <- soruyu kopyala
4    her cocukDugum <- yenisoru.cocukDugumler:
5     cocukDugum.uret()
6    dondur yenisoru
7   }
8  }
```

$P(x) = x^2 + 3x - 7 \quad 5[P(x)^2] = ?$ problemi için üretilen benzer problemler Tablo 6.2'de gösterilmiştir.

Tablo 6.2. Üretim örnekleri

$P(x) = x^2 + 3x - 7$	$5[P(x)^2] = ?$
$Q(x) = x^3 + 4x + 10$	$7Q(x) = ?$
$R(x) = x^2 + 3x + 6$	$3[R(x)^2] = ?$
$P(x) = x^4 + x - 2$	$P(x)^3 = ?$

Bir polinom probleminin benzerleri üretildiğinde, sonuçlar kullanıcıya gösterilmeden önce üretilen problemlerin doğruluğu kontrol edilir. [51] Eğer elde edilen problemde yer alan polinom ifadeleri polinom olma şartlarını sağlamıyorsa, bu problem kullanıcıya gösterilmez. Örneğin, üretilen problemde yer alan polinom ifadelerinden birinde dahi üssü negatif olan bir değişken varsa, bu problem kullanıcıya gösterilmez.

6.2. Sayı Sınıfı

Sayı sınıfına ait nesnelerin benzerlerini üretebilmek için geliştirilen programın sözde kodu Tablo 6.3'de gösterilmiştir.

Tablo 6.3. Sözde kod : sayı sınıfına ait nesnelerin benzerlerini üretme

```
1 Sinif Sayi <- MatematikNesnesi {
2   deger: Number
3   uret() {
4     deger <- (1 , deger+3) araliginda rastgele sayi uret
5   }
6 }
```

Örnek problemler için üretilen benzer problemler Tablo 6.4'de gösterilmiştir.

Tablo 6.4. Üretim örnekleri

$f(x) = 5x^2 + 3x$	$f(x) = 7x^4 + x$
$g(x) = x^3 + 4x + 10$	$g(x) = 2x^5 + 3x + 13$

6.3. Toplama Sınıfı

Toplama sınıfına ait nesnelerin benzerlerini üretebilmek için geliştirilen programın sözde kodu Tablo 6.5'de gösterilmiştir.

Tablo 6.5. Sözde kod : toplama sınıfına ait nesnelerin benzerlerini üretme

```
1  Sinif Toplama <- MatematikNesnesi {
2   uret() {
3    ustsinif.uret()
4    her cocukDugum <- cocukDugumler:
5     isaret <- (-1, 1) araliginda bir sayi uret
6     eger isaret kucuk 0 ise:
7      cocukDugum <- Cikarma(cocukDugum)
8     degilse: herhangi bir degisiklik yapma
9   }}
```

Örnek problemler için üretilen benzer problemler Tablo 6.6'da gösterilmiştir.

Tablo 6.6. Üretim örnekleri

$f(x) = 5x^2 + 3x$	$f(x) = -5x^2 - 3x$
$g(x) = x^3 + 4x + 10$	$g(x) = x^3 - 4x + 10$

6.4. Fonksiyon Sınıfı

Fonksiyon sınıfına ait nesnelerin benzerlerini üretebilmek için geliştirilen programın sözde kodu Tablo 6.7'de gösterilmiştir.

Tablo 6.7. Sözde kod : fonksiyon sınıfına ait nesnelerin benzerlerini üretme

```
1  Sinif Fonksiyon <- MatematikNesnesi {
2   isim: Metin
3   kullanilabilenIsimler: [Metin] = ["P", "Q", "R", "f", "g", "h"]
4   uret() {
5    ustsinif.uret()
6    eger bu fonksiyon icin isim degistirilmemisse:
7     isimler <- kullanilabilenIsimler listesinden daha once
          kullanilmayan ismleri bul
8     eger isim kucuk harfliyse:
9      isim <- isimler den kucuk harfli birini sec
10    degilse:
11     isim <- isimler den buyuk harfli birini sec
12    id icin isim ->> isimListesi
13   aksihalde:
14    isim <- id icin `isimListesi`nden ismi bul
15  }}
```

Örnek problemler için üretilen benzer problemler Tablo 6.8'de gösterilmiştir.

Tablo 6.8. Üretim örnekleri

$f(x) = 5x^2 + 3x$ $5[f(x)^2] = ?$	$g(x) = 5x^2 + 3x$ $5[g(x)^2] = ?$
$P(x) = x^3 + 4x + 10$ $der[P(x)] = ?$	$Q(x) = x^3 + 4x + 10$ $der[Q(x)] = ?$

6.5. Değişken Sınıfı

Değişken sınıfına ait nesnelerin benzerlerini üretebilmek için geliştirilen programın sözde kodu Tablo 6.9'da gösterilmiştir.

Tablo 6.9. Sözde kod : değişken sınıfına ait nesnelerin benzerlerini üretme

```
1  Sinif Degisken <- MatematikNesnesi {
2   isim: Metin
3   kullanilabilenIsimler = ["a", "b", "c", "d", "e"]
4   uret() {
5    eger isim "x" degilse:
6     eger bu degisken icin isim degistirilmemisse:
7      yeniIsim <- kullanilabilenIsimler icinden daha once
           kullanilmayan ismi bul
8      isim <- yeniIsim
9      id icin isim ->> `isimListesi`
10    degilse:
11     isim <- id icin `isimListesi`nden ismi bul
12   degilse:
13    herhangi bir degisiklik yapma
14  }
15 }
```

Örnek problem için üretilen benzer problem Tablo 6.10'da gösterilmiştir.

Tablo 6.10. Üretim örneği

$P(x) = x^a + b$	$P(x) = x^c + e$
$Q(x) = x + 2$	$Q(x) = x + 2$
$P(x) = Q(x)$	$P(x) = Q(x)$
$a + b = ?$	$c + e = ?$

6.6. Benzer Problemlerin Üretim Analizleri

Aşağıda bilgileri verilen bilgisayarda, geliştirilen çatı üzerinde polinom problem türleri için benzer problem üretme işlemleri yapılıp sonuçlar incelenmiştir.

MacBook Pro(Mid 2012)

İşlemci : 2.9 GHz Intel Core i7

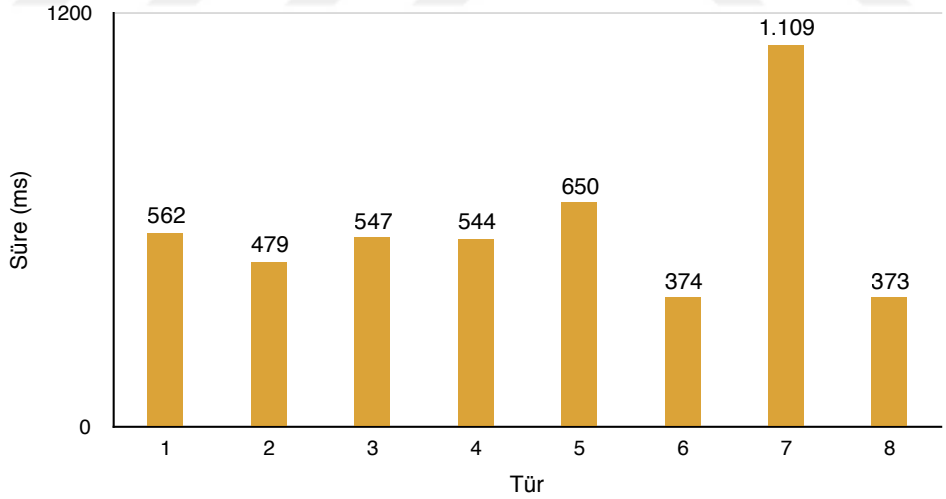
Bellek : 8 GB 1600 MHz DDR3

Sabit Disk : 250 GB SSD

İşletim Sistemi: MacOS Sierra(10.12.3)

6.6.1. Süre Analizi

Özellikleri belirtilen bilgisayarda, her bir türe ait bin adet soru rastgele üretilerek bu esnada geçen süreler ölçülmüştür. Bu sürelerin milisaniye cinsinden temsil edildiği grafik Şekil 6.1'de gösterilmiştir.

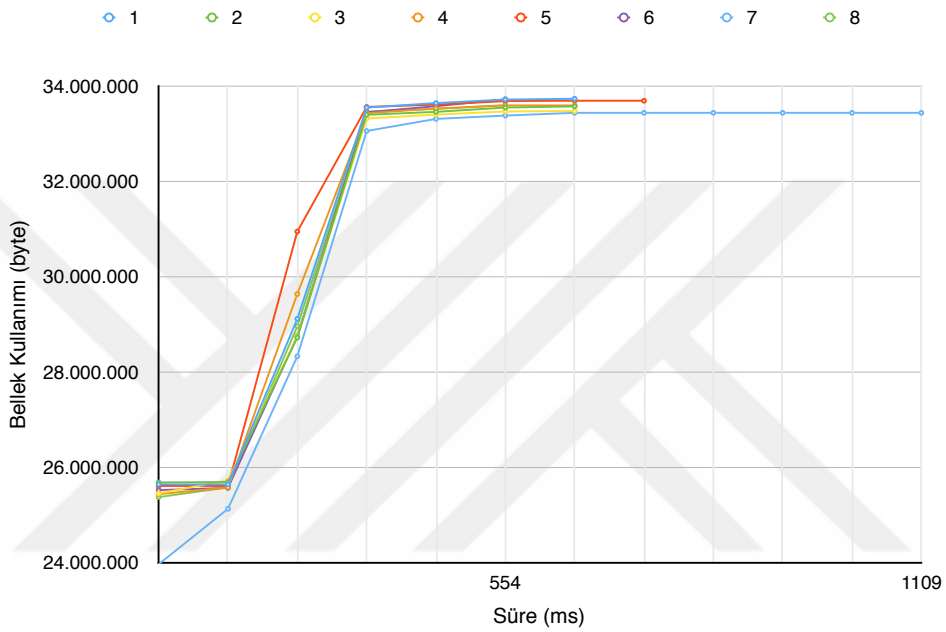


Şekil 6.1. Her türe ait 1000 sorunun üretim süreleri

Tüm problem türlerinin içerisinde değişime uygun olan ve olmayan kısımlar vardır. Yedinci problem türünün içerisinde değişime uygun olan kısımlar, diğer türlerin içerisinde değişime uygun olan kısımlardan fazla olduğu için, en uzun sürenin bu türde ölçülmesi beklenen bir sonuçtur.

6.6.2. Bellek Analizi

Özellikleri belirtilen bilgisayarda, her bir türe ait bin adet soru rastgele üretilerek bu süreç boyunca belirli aralıklarla, yaklaşık 100 milisaniye, bellek kullanımı miktarı ölçülmüştür. Problem üretme sürenin milisaniye, bellek kullanımı miktarının byte cinsinden temsil edildiği grafik Şekil 6.2'de gösterilmiştir.

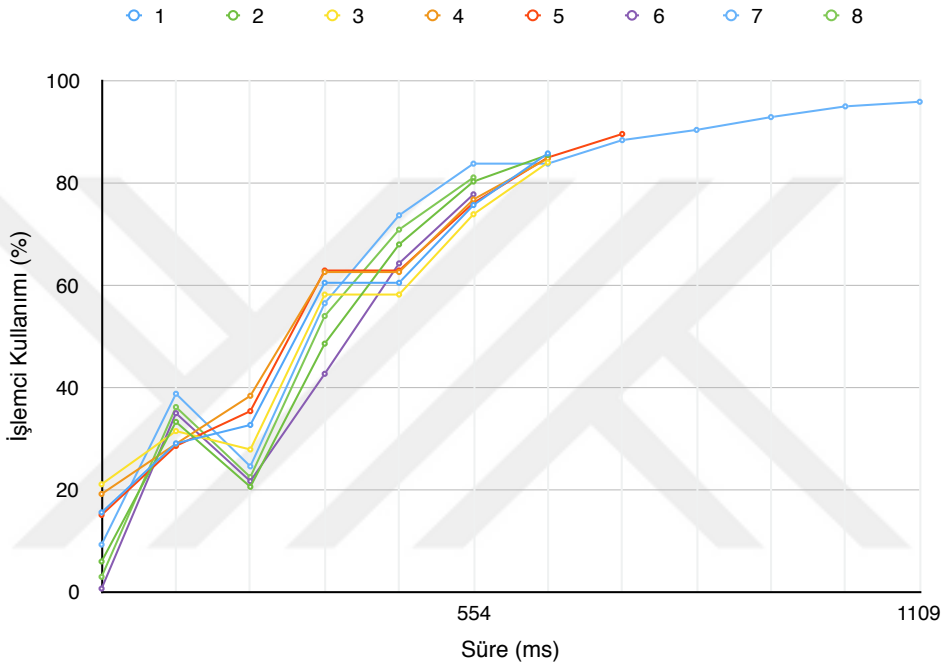


Şekil 6.2. Her türe ait 1000 sorunun üretim süreci boyunca kullanılan bellek miktarları

Problem türleri incelendiğinde, her problemin içerisinde benzer sayıda eleman olduğu görülür. Bu nedenle, problem türü ne olursa olsun, bir probleme benzer başka bir problem üretilirken bellekte tutulacak eleman sayıları birbirine yakındır. Bu durum problem üretme süreci boyunca kullanılan bellek miktarlarının yakın çıkmasının nedenidir.

6.6.3. Performans Analizi

Özellikleri belirtilen bilgisayarda, her bir türe ait bin adet soru rastgele üretilerek bu süreç boyunca belirli aralıklarla işlemci kullanımı yüzdesi ölçülmüştür. Problem üretme süresinin milisaniye, işlemci kullanımının % cinsinden temsil edildiği grafik Şekil 6.3'de gösterilmiştir.

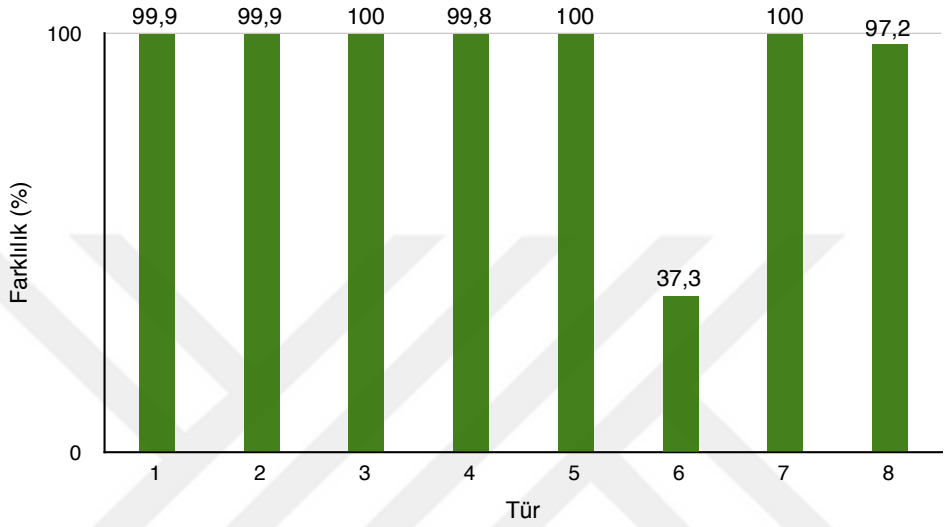


Şekil 6.3. Her türe ait 1000 sorunun üretim süreci boyunca kullanılan işlemci yüzdeleri

Yedinci problem türünün içerisinde, değişime uygun olan kısımlar diğer problem türlerinden fazla olduğu için, bu türe ait benzer problemler üretilirken, işlemci diğer türlere göre fazla kullanılır. Diğer yedi tür için ölçülen işlemci kullanım miktarları, problem türlerinde yer alan değişime uygun kısımlar benzer oranda olduğu için birbirine yakın çıkmıştır.

6.6.4. Üretilen Problemlerin Farklılığının Analizi

Özellikleri belirtilen bilgisayarda, her bir türe ait bin adet soru rastgele üretilerek bu soruların kaç tanesinin birbiriyle aynı olduğu tespit edilmiştir. Soruların birbirinden farklı olma durumunun % cinsinden temsil edildiği grafik Şekil 6.4'de gösterilmiştir.

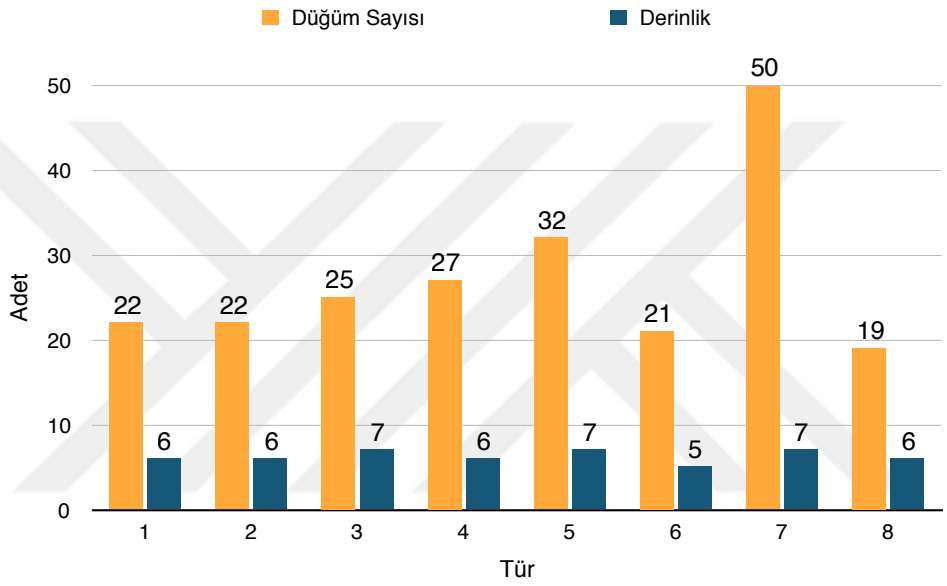


Şekil 6.4. Her türe ait üretilen 1000 sorunun farklılık %'si

Burada farklılık yüzdesi üretilen bin sorunun kaçının farklı olduğunu temsil etmektedir. Grafiğe bakıldığında birinci problem türünde üretilen bin sorunun dokuz yüz doksan dokuz tanesi farklıyken, altıncı problem türünde üç yüz yetmiş üç tanesinin farklı olduğu görülmektedir. Altıncı problem türünde değişime uygun olan kısımlar az olduğu için üretilen bin sorunun altı yüz yirmi yedi tanesi aynı çıkmıştır.

6.6.5. Düğüm Sayısı ve Derinlik Analizi

Özellikleri belirtilen bilgisayarda, her bir türe ait bin adet soru rastgele üretilerek bu soruların düğüm sayısı ve derinlik bilgileri kaydedilmiştir. Bu bilgilerin kendi aralarında toplanıp bine bölünmesiyle, her bir türe ait ortalama düğüm sayısı ve derinlik bilgisi elde edilmiştir. Üretilen soruların sahip olduğu ortalama düğüm sayısı ve derinlik bilgilerinin adet cinsinden temsil edildiği grafik Şekil 6.5'de gösterilmiştir.



Şekil 6.5. Her türe ait üretilen 1000 sorunun ortalama düğüm sayısı ve derinlik bilgileri

Bu grafik, benzer soru üretmede karmaşıklığa sebep olan durumları incelemek için önemlidir. Çalışmada önerilen ağaç yapısının, üretilen problemlerin türü ne olursa olsun derinliklerinin etkilenmemesini sağladığı görülmektedir. Ayrıca grafikte yer alan düğüm sayısı bilgileri, Şekil 6.1'de yer alan grafikteki benzer problem üretme süreleriyle birlikte değerlendirildiğinde, düğüm sayısının süreyi doğrudan etkilediği görülmektedir.

7. POLİNOM PROBLEMLERİNİN ADIM ADIM ÇÖZÜMÜ

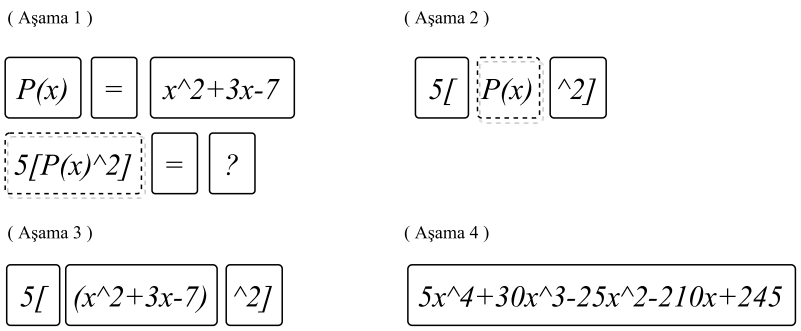
Problemlerin adım adım çözümü, sistemin çözücü bileşeni tarafından yapılır. Çözücü, sadeleştiriciden aldığı sadeleştirilmiş problemi, çözüm için önceden tanımlanan kurallar yardımıyla adım adım çözer. Sadeleştirme gerektiren çözüm adımlarında, çözücü ve sadeleştirici birlikte çalışır. Bu sayede, çözümün her adımında kullanıcıya en sade sonuç gösterilir.

Bu çalışmada problem çözmede geriye zincirleme ve cam kutu yaklaşımlarının benimsendiği bir uygulama çatısı geliştirilmiştir.

7.1. Geri Zincirleme

Her problem en az bir tanım ifadesi ve bir soru ifadesinden oluşur. Problem çözmede ileri ve geri zincirleme olmak üzere iki farklı yaklaşım vardır. Geri zincirleme, aranan soru ifadesini, verilen tanım ifadeleri yardımıyla bulma yaklaşımıdır. Bu amaçla ilk olarak sağ tarafında soru işareti olan eşitlik ifadesi bulunur. Ardından bulunan eşitliğin sol tarafındaki ifade ve bu ifadede yer alan bilinmeyenler belirlenir. Bu işlem tüm bilinmeyenler belirlenene kadar özyinelemeli olarak yapılır. Son olarak bilinmeyenler tanım ifadelerinden bulunur ve çözüm adımları sonucunda soru ifadesinin cevabına ulaşılır. Tüm bu aşamalar sırasında ihtiyaç duyulan yerlerde sadeleştirme işlemi yapılır.

$P(x) = x^2 + 3x - 7 \quad 5[P(x)^2] =?$ problemi için geriye zincirleme süreci Şekil 7.1'de gösterilmiştir.



Şekil 7.1. Geri zincirleme

Problemin geri zincirleme yaklaşımı ile çözümü için geliştirilen programın sözde kodu Tablo 7.1'de gösterilmiştir.

Tablo 7.1. Sözde kod : geri zincirleme algoritması

```
1  Sinif CozumTahtasi {
2   soru: Soru
3   bilinenler: [MatematikNesnesi]
4   bilinmeyenler: [MatematikNesnesi]
5   coz(){
6    soru.sadelestir()
7    her esitlik <- soru.Esitlikler
8     sagtaraf <- Esitlik.sagtaraf
9     soltaraf <- Esitlik.soltaraf
10    eger sagtaraf SoruIsareti sinifina uyeyse veya sagtaraf
          Fonksiyon sinifina uyeyse:
11      esitlik ->> bilinmeyenler
12    degilse:
13      esitlik ->> bilinenler
14   her bilinmeyen <- bilinmeyenler:
15    sagtaraf <- bilinmeyen.sagtaraf
16    soltaraf <- bilinmeyen.soltaraf
17    eger sagtaraf SoruIsareti sinifina uyeyse:
18     soltarafi sagtarafa kopyala
19    eger sagtaraf ve soltaraf Fonksiyon sinifina uyeyse:
20     her bililinen <- bilinenler:
21      bilinenSagtaraf <- bilinen.sagtaraf
22      bilinenSoltaraf <- bilinen.soltaraf
23      sagtaraf.degistir(bilinenSoltaraf, bilinenSagtaraf)
24   soru.sadelestir()
25   soru.ortakParantezeAl()
26  }
27 }
```

7.2. Cam Kutu

Problemler prosedürel ve kavramsal olmak üzere ikiye ayrılır. Prosedürel problemlerde verilen tanım ifadeleri yardımıyla soru ifadesinin cevabı bulunmaya çalışılır. Her problem kendine özgü çözüm adımlarına sahiptir. Bu problemlerin çözümünde başarılı olmak isteyenler, tanım ifadelerini doğru çözüm adımında kullanabilmelidir. Bilgisayar destekli öğrenme sistemleri geliştirilirken bu durum dikkate alınmalıdır. Kara kutu olarak bilinen yaklaşımda, arayüz yardımıyla alınan bir soru için sadece sistemin bulduğu cevap gösterilmektedir. Cam kutu yada beyaz kutu olarak bilinen yaklaşımda ise, arayüz yardımıyla alınan bir sorunun tüm çözüm adımları açıkça gösterilmektedir.

7.3. Çözüm Animasyonu

$P(x) = x^2 + 3x - 7 \quad 5[P(x)^2] =?$ problemi için, geliştirilen uygulama çatısı yardımıyla elde edilen çözüm aşamalarının bazıları Tablo 7.2'de gösterilmiştir.

Tablo 7.2. Çözüm animasyonu

Sahne	Sonuç
#2	$5 * (1 * (x^2)^2 * (3 * x - 7)^0 + 2 * (x^2)^1 * (3 * x - 7)^1 + 1 * (x^2)^0 * (3 * x - 7)^2)$
#6	$(1 * x^(2 * 2) * 1) * 5 + (2 * (x^2)^1 * (3 * x - 7)^1) * 5 + (1 * (x^2)^0 * (3 * x - 7)^2) * 5$
#12	$5 * x^(2 * 2) + (((3 * x) * 2 + -7 * 2) * x^2) * 5 + (1 * (x^2)^0 * (3 * x - 7)^2) * 5$
#16	$5 * x^(2 * 2) + (((3 * x) * 2) * 5) * x^2 + ((-7 * 2) * 5) * x^2 + (1 * 1 * (1 * ((3 * x))^2 * (-7)^0 + 2 * ((3 * x))^1 * (-7)^1 + 1 * ((3 * x))^0 * (-7)^2)) * 5$
#35	$5 * x^4 + 30 * x^(2 + 1) - 70 * x^2 + 5 * 3^2 * x^2 + ((2 * (3 * x) * (-7)^1) * 1) * 5 + ((1 * ((3 * x))^0 * (-7)^2) * 1) * 5$
#52	$5 * x^4 + 30 * x^(2 + 1) - 70 * x^2 + 5 * 3^2 * x^2 - 210 * x + 245$
#56	$30 * x^3 + 5 * x^4 + 45 * x^2 - 70 * x^2 - 210 * x + 245$
#58	$5 * x^4 + 30 * x^3 - 25 * x^2 - 210 * x + 245$

7.4. Problemlerin Çözüm Analizleri

Aşağıda bilgileri verilen bilgisayarda, geliştirilen çatı üzerinde polinom problem türleri için adım adım problem çözme işlemleri yapılıp sonuçlar incelenmiştir.

MacBook Pro(Mid 2012)

İşlemci : 2.9 GHz Intel Core i7

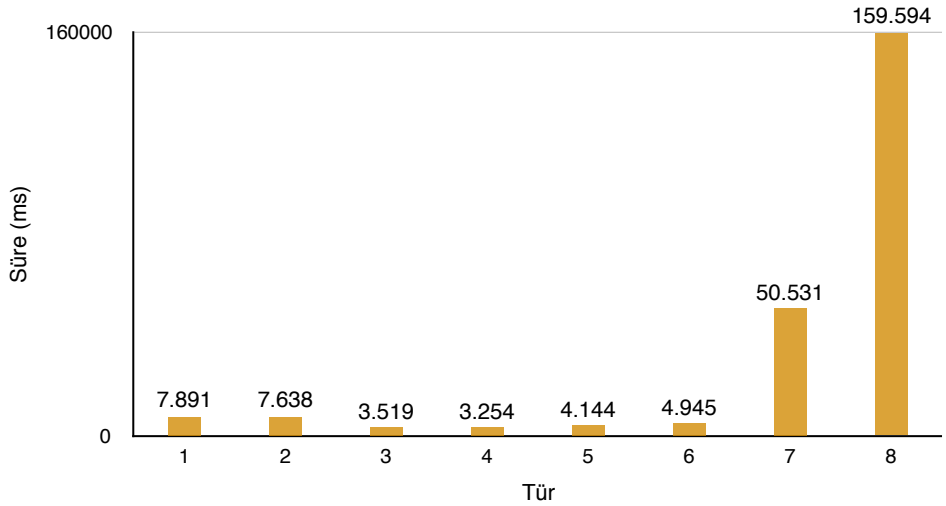
Bellek : 8 GB 1600 MHz DDR3

Sabit Disk : 250 GB SSD

İşletim Sistemi: MacOS Sierra(10.12.3)

7.4.1. Süre Analizi

Özellikleri belirtilen bilgisayarda her bir türe ait 1000 soru rastgele üretilip ardından adım adım çözülerek geçen süre hesaplanmıştır. Çözüm süresi, tüm süreden üretim süresinin çıkarılmasıyla tespit edilmiştir. Çözüm süresinin milisaniye cinsinden temsil edildiği grafik Şekil 7.2'de gösterilmiştir.

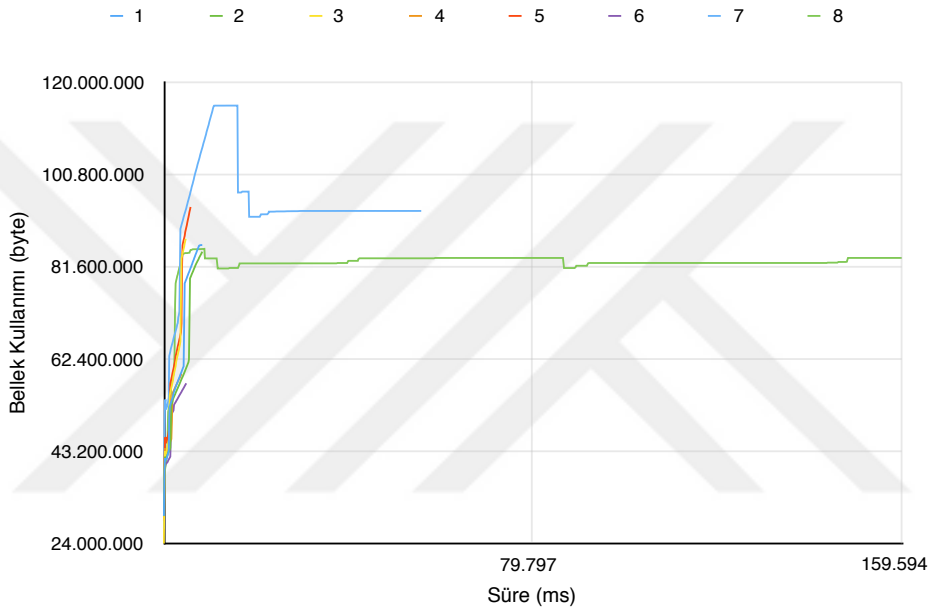


Şekil 7.2. Her türe ait 1000 sorunun adım adım çözüm süreleri

Sekizinci problem türü binom açılımı yapmayı gerektirdiği için, diğer yedi soru türüne göre daha fazla çözüm ve sadeleştirme adımı içerir. Bu yüzden en uzun süre, bu problem türünde ölçülmüştür.

7.4.2. Bellek Analizi

Bellek kullanımını ölçebilmek için öncelikle özellikleri belirtilen bilgisayarda her bir türe ait 1000 soru rastgele üretilip JSON verisi olarak kaydedilmiştir. Bu sorular adım adım çözülerek geçen süre boyunca bellek kullanımı miktarları ölçülmüştür. Problemlerin adım adım çözülme süresinin milisaniye, bellek kullanımının byte cinsinden temsil edildiği grafik Şekil 7.3'de gösterilmiştir.

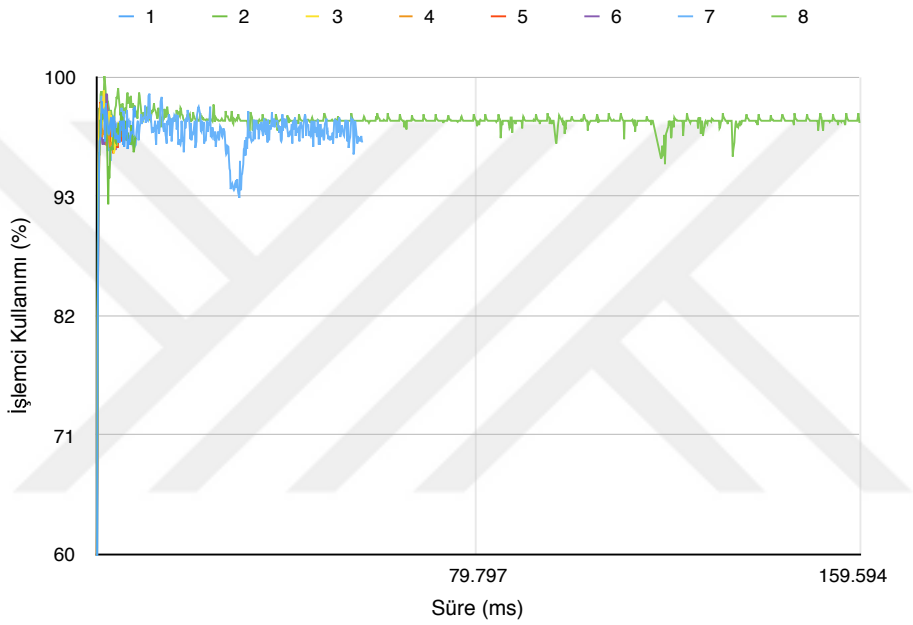


Şekil 7.3. Her türe ait 1000 sorunun çözüm süreci boyunca kullanılan bellek miktarları

Yedinci problem türü en fazla değişken içeren problem türü olduğu için, en fazla bellek kullanımı bu türde ölçülmüştür. Diğer problem türlerinin bellek kullanım miktarları ve süreleri birbirine yakındır. Sadece sekizinci problem türünün çözüm süresi uzun olduğu için, bellek kullanım miktarı diğer altı türle benzer olsada, belleği kullanma süresi diğer türlerin hepsinden fazladır.

7.4.3. Performans Analizi

İşlemci kullanımını ölçebilmek için ilk olarak özellikleri belirtilen bilgisayarda her bir türe ait 1000 soru rastgele üretilip JSON verisi olarak kaydedilmiştir. Daha sonra bu sorular adım adım çözülerek geçen süre boyunca işlemci kullanımı miktarları ölçülmüştür. Problemlerin adım adım çözülme süresinin milisaniye, işlemci kullanımının % cinsinden temsil edildiği grafik Şekil 7.4'de gösterilmiştir.

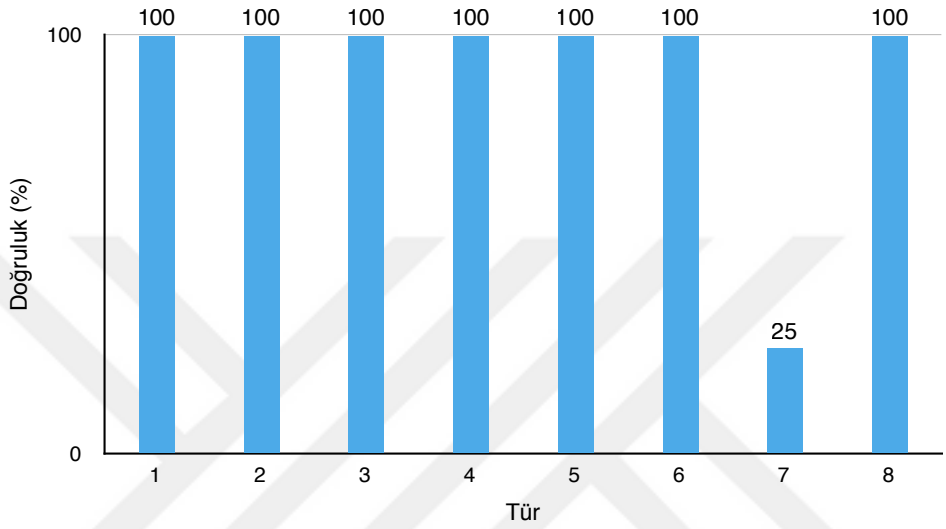


Şekil 7.4. Her türe ait 1000 sorunun çözüm süreci boyunca kullanılan işlemci yüzdeleri

Tüm problem türlerinin çözüm esnasında kulandıkları işlemci miktarları benzerdir. Sekizinci ve yedinci problem türleri, diğer türlere göre daha fazla çözüm ve sadeleştirme adımı içerdikleri için bu iki türün işlemciyi kullanma süreleri diğer türlerden fazladır.

7.4.4. Doğruluk Analizi

Özellikleri belirtilen bilgisayarda, her bir türe ait bin adet soru rastgele üretilip çözülerek, bu soruların kaç tanesinin çözümünün doğru olduğu tespit edilmiştir. Soruların doğru çözülme durumunun % cinsinden temsil edildiği grafik Şekil 7.5'de gösterilmiştir.



Şekil 7.5. Her türe ait çözülen 1000 sorunun doğruluk %'si

Burada doğruluk yüzdesi üretilen bin sorunun kaçının çözümünün doğru olduğunu temsil etmektedir. Grafiğe bakıldığında yedinci problem hariç tüm problem türlerinde üretilen bin sorunun ve çözümlerinin doğru olduğu görülmektedir. Yedinci problem türünde ise, üretilen bin sorunun dört yüz on dokuz tanesi yanlış üretilmiş, üç yüz yirmi yedi tanesi yanlış çözülmüş, yalnız iki yüz elli dört tanesi için doğru çözüm bulunmuştur. Bunun sebebi yedinci problem türünde üretme kısıtlarının doğru ayarlanamamasıdır.

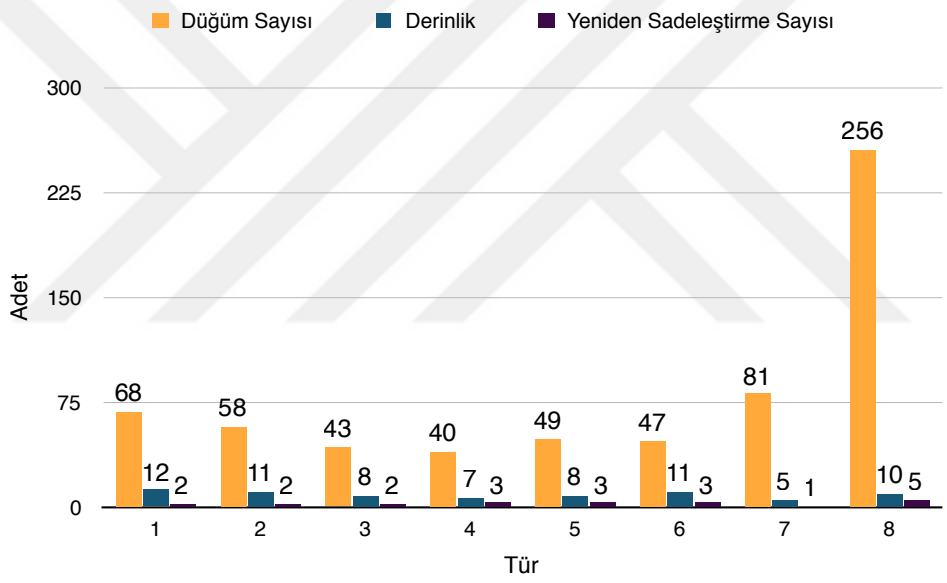
$$P(x) = (a-1)x^3 - bx^2 \quad Q(x) = dx^3 + cx^2 \quad P(x) = Q(x) \quad a+b+c+d = ?$$

$$P(x) = x^3 - bx^2 + 4x \quad Q(x) = 3x^2 - dx \quad P(x) = Q(x) \quad b++d = ?$$

Yukarıda yanlış üretilmiş soru örnekleri verilmiştir. Burada ilk soru için bir çözüm bulunamamaktadır. İkinci soruda ise soru yanlış üretildiği için x sıfır olarak bulunmakta ve sorunun çözümü yanlış olmaktadır. Yedinci problem türünde soru üretim aşamasında sorunun doğru üretilmesi sağlanamadığı için, çözüm aşamasında düşük bir doğruluk oranı elde edilmiştir.

7.4.5. Düğüm Sayısı, Derinlik ve Sadeleştirme Analizi

Düğüm sayısı, derinlik bilgisi ve sadeleştirme tekrar sayısı gibi bilgilere ulaşabilmek için öncelikle özellikleri belirtilen bilgisayarda her bir türe ait bin adet soru rastgele üretilip JSON verisi olarak kaydedilmiştir. Ardından bu sorular adım adım çözülerek çözüm sırasında ulaştıkları maksimum düğüm sayısı, maksimum derinlik bilgileri ve çözüm boyunca kaç kere sadeleştirmeye uğradıkları bilgisi kaydedilmiştir. Bu bilgilerin toplanıp bine bölünmesiyle, her bir türün ortalama maksimum düğüm sayısı, maksimum derinlik bilgisi ve etkilendiği ortalama sadeleştirme sayısı elde edilmiştir. Bu bilgilerin adet cinsinden temsil edildiği grafik Şekil 7.6'da gösterilmiştir.



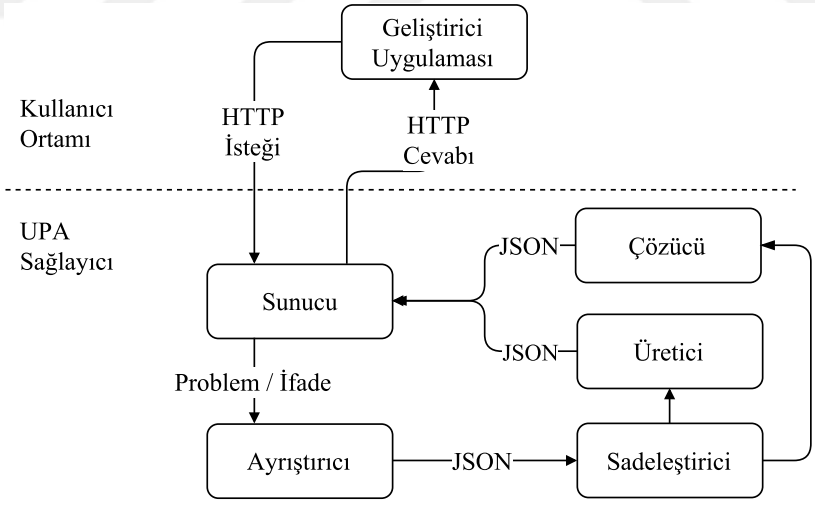
Şekil 7.6. Her türe ait 1000 sorunun çözüm süreleri boyunca ulaştığı ortalama maksimum düğüm, derinlik ve etkilendiği sadeleştirme adedi

Çözülen problemin türü ne olursa olsun, problemi temsil eden ağacın derinliği belirli bir değeri aşmamaktadır. Bu duruma, Şekil 2.2'de önerilen soyut sözdizim ağacı yapısı sebep olmaktadır. Çözüm süresi fazla olan problem türlerinin, düğüm sayılarının ve yeniden sadeleştirme sayılarının fazla olduğu grafiklerden görülmektedir.

8. ENTEGRASYON

Tez kapsamında sunulan uygulama çatısının, geliştiriciler tarafından üretilen farklı uygulamalara entegrasyonun sağlanabilmesi için, verilerin uygulamalar arasında kayıpsız bir şekilde aktarılabilmesi gerekir. Bu durum uygulamaların geliştirildiği ortamdan veya dilden bağımsız bir veri değişim formatı kullanma ihtiyacını ortaya çıkarır. Yapılan çalışmada, bu ihtiyaç doğrultusunda uygulamalar arası veri değişimi için JSON formatı kullanılmıştır. Çünkü modern yazılım dillerinin tamamı kendi JSON kütüphanelerine sahiptir. Bu durum JSON formatındaki verilerin farklı platformlara özellik kaybı olmadan iletilmesini garanti eder. Yapılan çalışmada, basit metin türündeki JSON istek verileri, HTTP isteğiyle geliştirilen uygulama çatısına iletilir. Geliştirilen çatı, gelen isteğe uygun sonucu üretir. Sonuç, HTTP cevabıyla isteği yapan uygulamaya iletilir.

Çalışmada geliştirilen uygulama programlama arayüzünün çalışma süreci Şekil 8.1'de gösterilmiştir.



Şekil 8.1. UPA genel yapısı

8.1. Uygulama Programlama Arayüzü (UPA)

Uygulama Programlama Arayüzü, bir uygulamanın farklı bir uygulamada kendisi için tanımlanmış işlevleri kullanabilmesini sağlayan bir tanım bütünüdür. Uygulama programlama arayüzleri, farklı yeteneklere sahip uygulamaların bir araya getirilerek daha gelişmiş yeni uygulamaların ortaya çıkmasına olanak sağlar. Tez kapsamında geliştirilen uygulama çatısının, farklı uygulamalar ile entegrasyonu için uygulama programlama arayüzü standardı oluşturulmuştur.

Geliştirilen UPA, uygulama çatısının bazı işlevlerinin farklı uygulamalar tarafından kullanabilmesini sağlamaktadır. İşlevlere ulaşmak için geliştirilen bir adres standardı kullanılmaktadır. İstekler, işlev adını ve istek türünü bulunduran adreslere, istek gövdesine isteğe ilişkin ifadeler eklenerek gönderilir.
İstek adresleri `http://adres/işlev_adı/[?t=(json | text)]` şeklindedir. İşlev adı uygulanmak istenen işlevi belirtir. Ayrıca isteğin ve sorgunun türü "t" parametresi ile verilebilir. Eğer "t" parametresi json olarak belirlenirse istek ve cevaplar json formatında, text olarak belirlenirse basit metin formatında olmalıdır.

Geliştirilen uygulama programlama arayüzüne yapılan bazı istekler ve karşılığında uygulama çatısından alınan cevaplar Tablo 8.1'de gösterilmiştir.

Tablo 8.1. Uygulama çatısına yapılan bazı istekler ve alınan cevaplar

Erişim Noktası	**İstek**	**Cevap**
/generate/?t=text	$f(x) = (x+1)^2 + x$	$g(x) = (3x+4)^4 + 2x$
/simplify/?t=text	$f(x) = (x+1)^2 + x$	$f(x) = x^2 + 3x + 1$

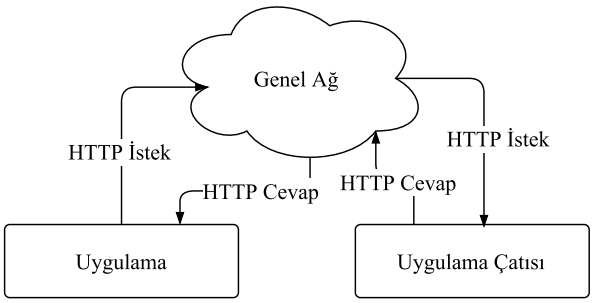
8.2. Entegrasyon Yaklaşımları

Geliştiricilere, tez kapsamında geliştirilen uygulama çatısını, iki farklı yaklaşımla kendi uygulamalarına ekleyebilme imkanı sağlanmaktadır. Birinci yaklaşım, herhangi bir ek kurulum yapmadan, varolan çevrimiçi UPA'yı kullanarak, istek gönderip cevap almaları ve bu cevapları kendi uygulamaları içinde kullanmalarıdır. İkinci yaklaşım ise, çatının tamamını kendi geliştirdikleri uygulama içerisine ayrı bir modül olarak eklemeleri ve çevrimiçi ortama bağımlı kalmadan sunulan UPA'yı kullanmalarıdır.

8.2.1. Çevrimiçi UPA Yardımıyla Entegrasyon

Çevrimiçi entegrasyon yöntemi olarak UPA kullanımı geliştiriciler için oldukça kolay ve yaygın kullanılan bir yöntemdir. Bu yöntem yardımıyla geliştiriciler farklı bir uygulamanın yeteneklerini ve verilerini kendi uygulamalarına kolaylıkla ekleyebilirler. Bu işlem belirtilen UPA standartlarına sahip çevrimiçi bir sunucuya istek gönderilerek yapılır. Geliştirilen uygulamada, genel ağ üzerinden önceden belirlenen adres ve standartlara uygun HTTP istekleri alınır ve isteklere uygun HTTP cevapları karşılık olarak verilir.

Çevrimiçi entegrasyon yaklaşımı Şekil 8.2'de gösterilmiştir.

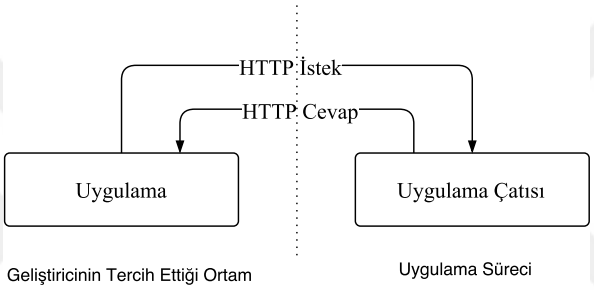


Şekil 8.2. Çevrimiçi UPA yardımıyla entegrasyon

8.2.2. Bileşen Olarak Entegrasyon

Çalışma kapsamında geliştirilen uygulama çatısının tamamı, açık kaynaklı araçlar kullanılarak geliştirilmiştir. Bu geliştirme biçimi, geliştiricilerin mevcut çatıyı kendi uygulamalarına bir bileşen olarak ekleyebilmelerine olanak sağlar. Geliştiriciler, çatının tamamını kendi uygulamalarına entegre ederek, farklı bir süreçte çalışmasını sağlayabilir. Ayrıca bu şekilde entegrasyon yapsalarda, UPA standartlarını ve adreslerini kullanarak, çevrimiçi UPA'ya istekte bulunup cevap almaya devam edebilirler.

Bileşen olarak entegrasyon yaklaşımı Şekil 8.3'de gösterilmiştir.



Şekil 8.3. Bileşen olarak entegrasyon

9. SONUÇLAR

Bu çalışmada çevrimiçi ve çevrimdışı çalışabilen uygulamaların kullanabileceği, polinom problemleriyle ilgili simgesel hesaplamalar için bir uygulama çatısı ve bu çatı kullanılarak geliştirilmiş MathBox isimli bir web uygulaması sunulmuştur. Kullanıcılar geliştirilen uygulama yardımıyla, farklı türdeki polinom problemlerinin benzerlerini üretebilir ve polinom problemlerini adım adım çözebilir. İlgilendikleri probleminin farklı polinomlar için ürettiği sonuçları gören kullanıcılar, problemi temsil eden doğru polinomları kolaylıkla bulabilir. Uygulama, polinom problemlerinin simgesel hesaplamasını kullanıcı bilgisayar kaynaklarından faydalanacak şekilde istemci tarafında yaptığı için, işlemler çok kısa sürmektedir. Bu durum hem uygulamanın bulunduğu sunucunun yükünü azaltır hem de bu sunucunun aynı anda cevap verebileceği kullanıcı sayısını artırır.

Her bir türe ait bin adet benzer problem üretilirken elde edilen ortalama derinlik, ortalama düğüm sayısı, toplam süre, toplam bellek kullanımı, toplam işlemci kullanımı ve farklılık oranı bilgileri Tablo 9.1'de gösterilmiştir.

Tablo 9.1. Benzer problem üretme: özet

Tür	Derinlik	Düğüm s.	Süre (ms)	Bellek(byte)	İşlemci(%)	Farklılık(%)
1	6	22	562	33.734.656	85.8	99.9
2	6	22	479	33.574.912	85.5	99.9
3	7	25	547	33.476.608	84	100
4	6	27	544	33.595.392	84.8	99.8
5	7	32	650	33.693.696	89.6	100
6	5	21	374	33.722.368	77.8	37.3
7	7	50	1109	33.439.744	95.9	100
8	6	19	373	33.599.488	81.1	97.2

Grafikten, problemin düğüm sayısının süreyi ve işlemci kullanımını doğrudan etkilediği görülmektedir. Ayrıca değişken sayısının farklılık oranını etkilediği anlaşılmaktadır.

Her bir türe ait bin adet problem adım adım çözülürken elde edilen ortalama derinlik, ortalama yeniden sadeleştirme sayısı, ortalama düğüm sayısı, toplam süre, toplam bellek kullanımı, toplam işlemci kullanımı ve doğruluk oranı bilgileri Tablo 9.2'de gösterilmiştir.

Tablo 9.2. Adım adım problem çözme: özet

Tür	Derinlik	Y. S. S.	Düğüm s .	Süre (ms)	Bellek(byte)	İşlemci(%)	Doğruluk(%)
1	12	2	68	7891	86.130.648	95.8	100
2	11	2	58	7638	84.742.144	93	100
3	8	2	43	3519	87.445.504	96	100
4	7	3	40	3254	59.305.984	95.6	100
5	8	3	49	4144	94.056.448	96	100
6	11	3	47	4945	57.524.254	95.6	100
7	5	1	81	50531	115.163.136	95.4	25.4
8	10	5	256	159594	85.340.160	100	100

Grafikten, derinliğin aksine problemin düğüm sayısının süreyi ve işlemci kullanımını doğrudan etkilediği görülmektedir. Ayrıca düğüm sayısı fazla olan problemlerin çözüm adımları boyunca daha fazla sadeleştirme işlemine uğradıkları anlaşılmaktadır. Yedinci problem türü dışındaki problem türlerinde rastgele üretilen bin problem doğru olduğu için, adım adım çözümlerinin doğruluk oranı da %100 çıkmıştır.

Çalışma kapsamında geliştirilen uygulama çatısı, sunulan uygulama programlama arayüzü sayesinde, geliştiricilerin kendi mobil, web veya masaüstü uygulamalarına kolaylıkla entegre edilebilecek niteliktedir. Bu durum, geliştirilen uygulamalara simgesel hesaplama yapabilme kabiliyeti kazandıracaktır. Böylece geliştirilen uygulama çatısı, polinomlar için simgesel hesaplama gerektiren matematiksel modellerin geliştirilmesinde, Lagrange ve Newton gibi sayısal çözümleme yöntemlerinin mühendislik problemlerine uygulanması süreçlerinde, yardımcı araç olarak kullanılabilecektir.

Yapılan çalışma sırasında yazılan üç adet bildiri, ICITS 2016 [52], MME 2017 [53] ve ICAT 2017 [54] konferanslarında sözlü olarak sunulmuş ve ilgili konferansların bildiri kitapçıklarında basılmıştır.

10. GELECEK ÇALIŞMALAR

Kullanıcıdan alınan polinom problemlerinin benzerlerinin üretilmesi veya adım adım çözülmesi sırasında yapılan sadeleştirme işlemlerinin hızı, problemlerin düğüm sayısının artmasıyla düşmektedir. Karmaşık bir işlemi oluşturan her bir alt işlemin, aynı anda farklı işlemciler tarafından yapılması paralel hesaplama olarak adlandırılır. Paralel hesaplama, karmaşık işlemlerin tamamlanması için gereken süreyi ciddi bir şekilde azaltır. İlerleyen çalışmalarda, mevcut uygulama çatısına paralel hesaplama yeteneği kazandırılması, bu sayede daha hızlı bir simgesel hesaplama ortamı elde edilmesi amaçlanmaktadır.

Çalışma kapsamında geliştirilen uygulama çatısı yapay zeka içermemektedir. Bu nedenle çatı, problem çözme seviyesini ölçmek yada bu seviyelerin ilerleme tespitini yapmak gibi yapay zeka gerektiren uygulamalar geliştirilirken kullanılmaya uygun değildir. İlerleyen çalışmalarda, uygulama çatısına yapay zeka yeteneği kazandırılarak bu sorunun aşılması hedeflenmektedir.

11. KAYNAKLAR

1. Elbourn, R. D., ve Ware, W. H., The evolution of concepts and languages of computing, Proceedings of the IRE, 50, 5 (1962) 1059-1066.

2. Von Zur Gathen, J., ve Gerhard, J., Modern computer algebra, Cambridge university press, 2013.

3. Cohen, J. S., Computer algebra and symbolic computation: Mathematical methods, Universities Press, 2003.

4. Shirley, P., Ashikhmin, M., ve Marschner, S., Fundamentals of computer graphics, CRC Press, 2015.

5. Guo, L., Wanga, J., ve Wub, H., Application of Secret Sharing in XML Protection Mechanism, Procedia Computer Science, 107 (2017) 21-26.

6. Koblitz, N., Algebraic aspects of cryptography, Springer Science & Business Media, 2012.

7. Zhou, J., Bagnell, A., ve Mason, M. T., A Fast Stochastic Contact Model for Planar Pushing and Grasping: Theory and Experimental Validation, Robotics: Science and Systems, 2017, Massachusetts-USA.

8. Smith, S., Digital signal processing: a practical guide for engineers and scientists, Newnes, 2013.

9. Wexler, J. D., A self-directing teaching program that generates simple arithmetic problems, Computer Sciences Technical Report, 19 (1968).

10. Fateman, R. J., Code generation: evaluating polynomials, University of California, 2002, Berkeley.

11. Osorio, M. A., ve Cuaya, G., Hard problem generation for MKP, Sixth Mexican International Conference on (IEEE), 2005, Mexico.

12. Kumar, A. N., Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors, Technology, Instruction, Cognition and Learning (TICL) Journal, 4, 1 (2006).

13. Gulwani, S., Dimensions in program synthesis, Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming (ACM), 2010, Hagenberg-Austria.

14. Tillmann, N., ve de Halleux, J., Pex–white box test generation for. net, International conference on tests and proofs, 2008, Berlin, Springer, 134-153.

15. Sadigh, D., Seshia, S. A., ve Gupta, M., Automating exercise generation: A step towards meeting the MOOC challenge for embedded systems, Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education(ACM), 2012, Tampere-Finland.

16. Singh, R., Gulwani S., and Solar-Lezama, A., Automated feedback generation for introductory programming assignments, Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation(ACM), 2013, Seattle-Washington, 15-26.

17. Ahmed, U. Z., Gulwani, S., ve Karkare, A., Automatically Generating Problems and Solutions for Natural Deduction, 23th International Conference on Artificial Intelligence(IJCAI), Ağustos 2013, Beijing-China.

18. Nistal, M. L., Tato, J. G., Rodríguez, M. C., Gago, J. M. S., ve Sabucedo, L. M. A., e-Problems: An automatic approach to the generation of practical problems in e-learning, Tecnologias Aplicadas a la Ensenanza de la Electronica (Technologies Applied to Electronics Teaching-TAEE), Haziran 2014, Bilbao-Spain.

19. Shenoy, V., Aparanji, U., Sripradha K. ve Kumar, V., Generating DFA Construction Problems Automatically, Learning and Teaching in Computing and Engineering(LaTICE), Nisan 2016, Mumbai-India.

20. Bastani, O., Sharma R., Aiken A. ve Liang, P., Synthesizing Program Input Grammars, CoRR, 1608, 01723 (2016).

21. Karkare, A., ve Agarwal, N., ParseIT: A Question-Answer based Tool to Learn Parsing Techniques, SIGCSE '16 Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 2017, Memphis-Tennessee.

22. Kojima, K., Miwa, K., ve Matsui, T., Study on support of learning from examples in problem posing as a production task, Proceedings of the 17th International Conference on Computers in Education, Ağustos 2009, Hong Kong.

23. Kukelova, Z., Bujnak, M., ve Pajdla, T., Automatic generator of minimal problem solvers, Computer Vision–ECCV 2008, 2008, Berlin, 302-315.

24. Andersen, E., Gulwani, S., ve Popovic, Z., A trace-based framework for analyzing and synthesizing educational progressions, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems(ACM), 2013, USA.

25. Alvin, C., Gulwani, S., Majumdar, R., ve Mukhopadhyay, S., Automatic Synthesis of Geometry Problems for an Intelligent Tutoring System, CoRR, (2015).

26. Polozov, O., O'Rourke, E., Smith, A. M., Zettlemoyer, L., Gulwani, S., ve Popovic Z., Personalized Mathematical Word Problem Generation, IJCAI, Temmuz 2015, Arjantin.

27. Jurkovic, N., Diagnosing and correcting student's misconceptions in an educational computer algebra system, Proceedings of the 2001 international symposium on Symbolic and algebraic computation(ACM), Temmuz 2001, Canada.

28. Singh, R., Gulwani, S., ve Rajamani, S., Automatically Generating Algebra Problems, AAAI-12, Temmuz 2012, Toronto.

29. Brooker, R. A., The solution of algebraic equations on the EDSAC, Mathematical Proceedings of the Cambridge Philosophical Society, 48, 2 (1952) 255-270.

30. Muller, D. E., A method for solving algebraic equations using an automatic computer, Mathematical Tables and Other Aids to Computation, 10, 56 (1956) 208-215.

31. Goldstein, I., ve Papert, S., Artificial intelligence, language, and the study of knowledge, Cognitive Science 1, 1 (1977) 84-123.

32. Beeson, M. J., Logic and computation in MATHPERT: An expert system for learning mathematics, Computers and mathematics, (1989) 202-214.

33. www.helpwithmath.com/ MathXpert. 1 Mayıs 2017.

34. www.softmath.com/ ALGEBRATOR. 1 Mayıs 2017.

35. Joyner, D., Čertík, O., Meurer, A. ve Granger, B. E., Open source computer algebra systems: SymPy, ACM Communications in Computer Algebra, 45, 3 (2012) 225-234.

36. www.sympy.org/ SymPy. 1 Mayıs 2017.

37. maxima.sourceforge.net/ Maxima. 1 Mayıs 2017.

38. www.wolfram.com/mathematica/ WOLFRAM MATHEMATICA. 1 Mayıs 2017.

39. Cocke, J., Global common subexpression elimination, ACM Sigplan Notices., 5,7 (1970) 20-24.

40. Moses, J., Algebraic simplification a guide for the perplexed, Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, 1971, California-USA.

41. Hosangadi, A., Fallah, F., ve Kastner, R., Factoring and eliminating common subexpressions in polynomial expressions, Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design, 2004, San Jose - USA.

42. Malaquias, J. R., ve Lopes, C. R., Implementing a computer algebra system in Haskell, Applied Mathematics and Computation, 192, 1 (2007) 120-134.

43. Bailey, D. H., Borwein, J. M., ve Kaiser, A. D., Automated simplification of large symbolic expressions, Journal of Symbolic Computation, 60 (2014) 120-136.

44. 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.

45. www.ginac.de/ Ginac. 1 Mayıs 2017.

46. Aho, A. V., Lam, M. S., Sethi, R., ve Ullman J. D., Compilers, Principles, Techniques, Boston: Addison wesley, 1986.

47. www.javacc.org/ JavaCC. 1 Mayıs 2017.

48. www.json.org/ JSON. 1 Mayıs 2017.

49. developer.mozilla.org/en-US/docs/Web/JavaScript/ JavaScript. 1 Mayıs 2017.

50. Horner, W. G., A new method of solving numerical equations of all orders, by continuous approximation, Philosophical Transactions of the Royal Society of London, 109 (1819) 308-335.

51. Schwartz, J. T., Fast probabilistic algorithms for verification of polynomial identities, Journal of the ACM (JACM) 27, 4 (1980) 701-717.

52. Efendioğlu, S., ve Efendioğlu H. E., Developing Proficiency of Students Especially in the Algebra Problems Which Are Difficult for Them by Means of Customized Automatic Question Generation, 10th International Computer and Instructional Technologies Symposium(ICITS-2016), Mayıs 2016, Rize-Türkiye.

53. Efendioğlu, S., ve Efendioğlu H. E., An Alternative Computerized System for Step-by-Step Solution Analysis of Algebra Problems With Multiple Mathematical Expressions, International Workshop on Mathematical Methods in Engineering(MME-2017), Nisan 2017, Ankara-Türkiye.

54. Efendioğlu, S., Efendioğlu H. E., ve Pehlivan, H., A Computer Algebra System That Can Be Integrated Into Users' Working Environments to Support Symbolic Computation Operations, 5th International Conference on Advanced Technology & Sciences(ICAT'17), Mayıs 2017, İstanbul-Türkiye.

ÖZGEÇMİŞ

Seda EFENDİOĞLU, 1990 Trabzon TÜRKİYE doğumludur. İlk öğretiminin ilk dört yılını Mareşal Fevzi Çakmak İlköğretim Okulu'nda son dört yılını Pelitli 75. Yıl Cumhuriyet İlköğretim Okulu'nda ve lise eğitimini Tevfik Serdar Anadolu Lisesi'nde tamamlamıştır. 2009-2010 güz yarıyılında Karadeniz Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü'nde başladığı lisans eğitiminden 2012-2013 bahar yarıyılında mezun olmuştur. 2014-2015 güz yarıyılında yüksek lisansa başlayıp araştırma görevlisi olarak da görev aldığı Karadeniz Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü'nde eğitimini ve görevini sürdürmektedir. İyi düzeyde İngilizce, orta düzeyde Korece bilmektedir.