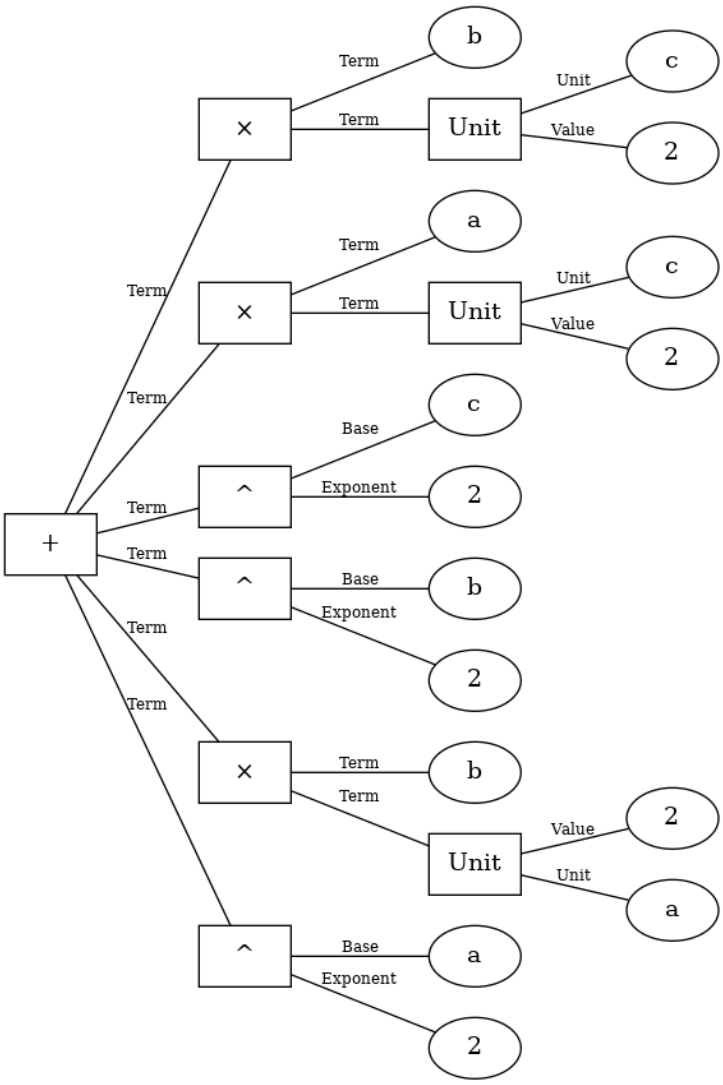
GAMLP: En symbolhanterare

Matstoms, Axel Janković, Luka Matstoms, Ivar

15 mars 2018



Abstract

The purpose of this project is to look at some of the techniques used by sites such as WolframAlpha to simplify algebraic expressions and solve particular kinds of equations. To research this we wrote an application in Python with some of the features of such sites. We chose to work with Python because of how quickly code can be written in Python and due to the fact everybody in the group knew Python before the project even started. During the span of this project we were able to develop an application which is able to take input and construct an abstract syntax tree which is then simplified. The application is able to handle many of our goals set in the beginning, including addition and multiplication of polynomials correctly, although not always fully simplified. The application is also able to solve polynomials up to the second degree. With time the application could easily be expanded to cover more aspects of what we set out to do such as polynomial division and solving polynomial equations of higher grades.ga

Keywords: Abstract syntax tree; Python; Algebraic Expressions

Innehåll

1 Inledning

Eftersom vi alla i gruppen delar en stor passion för programmering ville vi göra något som kombinerar programmering med ett naturvetenskapligt ämne. Vi övervägde flera olika projekt, bland annat en realtids fysikmotor, implementationer av matematiska funktioner (såsom sinus, exponenter) på hårdvarunära nivå samt idén som valdes: en symbolhanterare. En symbolhanterare, eller CAS, Computer Algebra System, är ett datorprogram som kan mata in, tolka, och lösa ekvationer, exempelvis $5x + 10 = 20$. En annan funktion som en symbolhanterare antar är förmågan att förenkla uttryck, exempelvis $(x + 2)(x - 3) \Leftrightarrow (x^2 - x - 6)$. Skillnaden mellan en vanlig miniräknare och en symbolhanterare är att symbolhanterare förstår algebra och kan samt kan hantera variabler, inte bara värden.

2 Teori

2.1 Abstrakt syntaxträd

Det är besvärligt för ett program att tolka ekvationer som är skrivna på vanlig form (e.x. $5x + 10 = 15$). Därför är det viktigt att ekvationen först byggs upp på ett sätt som är mer hanterbart för ett datorprogram. Så kallade abstrakta syntaxträd används oftast inom datavetenskap för att strukturera programmeringskod så att en kompilator kan omvandla källkoden till ett exekverbart program. Men på grund av likheterna mellan programmeringsspråkens syntax och matematiska ekvationer användes abstrakta syntaxträd i detta arbete.

2.1.1 Allmänt om träd

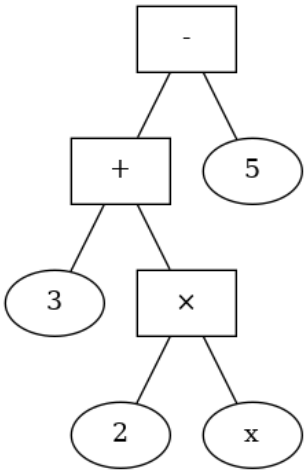
Inom datavetenskap beskrivs träd som en form av datastruktur för att representera hierarkisk data. Ett träd inom matematiken består av noder som är sammansatta på så vis att varje nod kan innefatta ett eller flera barn, som i sig också är noder. I det här dokumentet kallas barnen för barnnoder eller subnoder. Noder behöver dock ej innefatta några barn, och då kallas dessa noder för löv. Dock behöver varje nod anta ett värde. Detta kan vara t.ex. ett numeriskt värde (e.x. 5 eller 10), eller en operation (e.x. addition, multiplikation). Om en nod ej är ett barn till en annan nod betyder det att denna nod är högst upp i trädet. Denna typ av nod kallas för rotnod eller bara rot.

2.1.2 Implementation av syntaxträd

Trädet i sig utgår från en rotnod (eng. rootnode). I vårt fall är denna nod alltid ett likhetstecken (=), eftersom det är där ekvationen utgår ifrån. Precis som t.ex. en operatörsnod kan rotnoden anta två subnoder (I detta fall måste två noder antas, en för VL och en för HL). Därefter kan leden byggas upp av olika sorters noder.

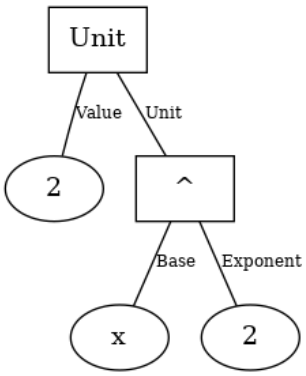
2.1.3 Noder

En nod måste anta ett värde. Detta kan uppnås genom att ge noden ett konkret värde (e.x. 5 eller 10), en konstant (e.x. π, e), funktion (e.x. log, sin) eller operator (e.x. $+$, $-$, $\times$, $\div$, $\hat{}$). Om noden är sådan att den kräver argument, dvs. kräver ett eller flera värden för att själv få ett värde ($+$ har ej ett värde utan två siffror på vardera sida av sig.) så kan noden själv anta noder (s.k. barnnoder, eng. childnodes) för att uppfylla sitt värde. Se Figur 1. Eftersom programmet ska kunna hantera ekvationslösning, måste



Figur 1: Visuell representation av $(2x + 3) - 5$ som ett abstrakt syntaxträd.

den okända variabeln kunna representeras (e.x. $5x + 10 = 20$. Här är x den okända variabeln). För att representera variabeln används en speciell typ av nod; så kallad enhetsnod. Denna nod antar ett värde (för koefficienten) och en representation av den okända variabeln; en så kallad enhet. Denna enhet består av själva variabeln och vilken grad som den representeras i. Se Figur 2 för en visuell representation av $2x^2$ som en enhetsnod.



Figur 2: $2x^2$ representerat som en enhetsnod med enheten x^2 och värdet 2.

2.2 Allmänt om förenkling

Operatörsnoder vars subnoder endast består av numeriska värden räknas helt enkelt ihop med den operatorn som noden representerar. Då kommer denna operatörsnod ersättas med en nummernod som antar det ihopräknade värdet.

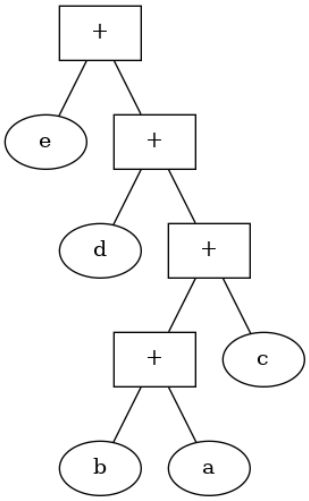
2.3 Homogena noder

Homogena noder är de noder där ordningen av subnoder ej spelar någon roll vilket gör att de kan lagras som en lista (array) och är inte limiterad i antal. Dock byggs homogena noder först upp av flera noder med bara två subnoder (Se 2.4 Inhomogena noder) eftersom det är så programmet tolkar inmatningen, se Figur 3.

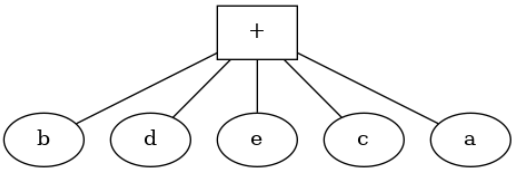
2.3.1 Allmän förenkling av homogena noder

Att förenkla noder som precis har tolkats av programmet (se Figur 3) är dock opraktiskt. Därför är det första steget i förenklingen att försumma staplade homogena noder av samma typ till en enda nod. Om noderna antar numeriska värden räknas de ihop till en subnod, medans variabler (x) helt enkelt lämnas som en subnod. Efter det läggs elementen in i noden en efter en och varje element kontrolleras om den kan slås ihop med en annan subnod. Detta gäller dock endast för noder med numeriska värden. Hur enhetsnoder (se 2.1.3 för förklaring av enhetsnod) hanteras är dock olika för varje typ av homogen nod. Därför förklaras detta i detalj i samband med varje typ av nod.

Två enhetsnoder med samma enhet kan slås ihop genom att addera värderna.



Figur 3: $(a + b + c + d + e)$ tolkat av programmet.



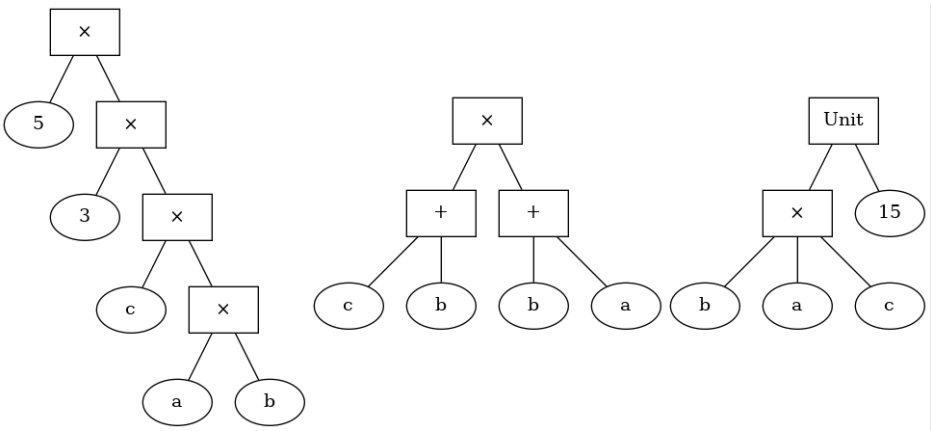
Figur 4: $(a + b + c + d + e)$ efter första steget av förenkling av homogen nod

2.3.2 Additionsnod

Förenklingen av en additionsnod funkar genom att fylla på en tom lista med värden från en additionsnods barnnoder och kollar om en nod i barnnodslistan kan läggas ihop med en nod i den nya listan. T.ex. om en additionsnod består av bara siffernoder så kommer dessa adderas. Om flera enhetsnoder med samma enhet förekommer så kommer deras koefficienter summeras (se Figur 6). En potensnod kan slås ihop med en annan nod om basen på potensnoden är samma som den andra noden, isåfall skapas en ny potensnoden med exponenten ökad med ett.

2.3.3 Multiplikationsnod

Efter de generella förenklingarna av homogena noder (Se 2.3.1 Allmän förenkling av homogena noder) separeras additionsnoder från övriga noder. Alla kombinationer av dessa separerade additionsnoders subnoder multipliceras med övriga noder och adderas ihop. Enhetsnoder kombineras genom att skapa en ny enhetsnod vars koefficient består av produkten av de ori-



Figur 5: $(a \times b \times c \times 3 \times 5)$ förenklas i tre steg, a,b,c lyfts upp till samma nod och talen separeras till sin egen nod, tillslut räknas talen ihop.

ginala enhetsnodernas koeficienter och vilken grad den nya enhetsnoden representeras i bestäms av hur många enhetsnoder det från början var. (e.x. $5x \times 10x = 50x^2$). Enhetsnoder kan också kombineras med övriga noder genom att de övriga värdena multipliceras in i enhetsnodens värde (Se 2.1.3 Noder).

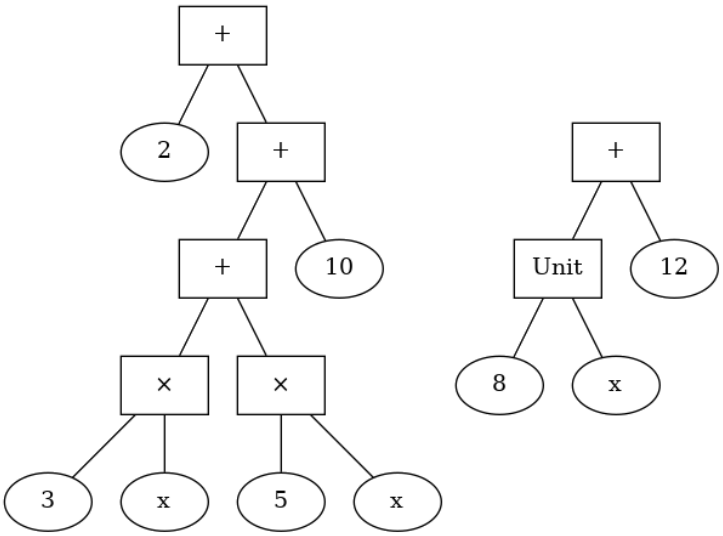
2.4 Inhomogena noder

Inhomogena noder genomför inhomogena operationer; dvs. operationer som där ordningen av subnoderna spelar roll. (e.x. $a - b \neq b - a$). Därmed kan inhomogena noder ej kombineras rekursivt på samma sätt som homogena noder kan. (Se 2.3.1 Allmän förenkling av homogena noder) Detta medger också att det inte finns ett allmänt sätt att förenkla inhomogena noder.

2.4.1 Potensnod

Potensnoder innehåller två barnnoder, basen och exponenten. En potensnod med en additionsnod i basen, exempelvis $(a + b + c)^3$, expanderas med hjälp av multinomialsatsen. Som man kan ana på namnet liknar multinomialsatsen binomialsatsen. Skillnaden är att multinomialsatsen är ett mer generaliserat sätt att beskriva utveckling av polynom, och är alltså ej begränsad till endast binom. Satsen lyder

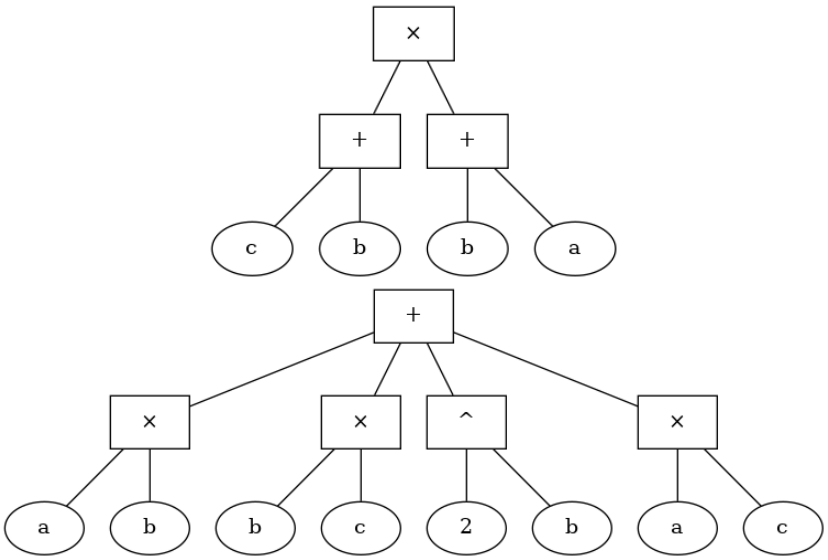
$$(a_1 + a_2 + \ldots + a_m)^n = \sum_{k_1+\cdots+k_m=n} \binom{n}{k_1, k_2, \ldots, k_m} a_1^{k_1} \cdot \ldots \cdot a_m^{k_m}$$



Figur 6: $5x + 3x + 10 + 2$ förenklas till $8x + 12$

Den behöver dock lite förklaring. Med $\sum_{k_1+\cdots+k_m=n}$ menas alla möjliga kombinationer med återläggning av exponenter i $a_1^{k_1} \cdot a_2^{k_2} \cdot \ldots \cdot a_m^{k_m}$ där $\sum_{i=0}^{m} k_i = n$ uppfylls. Vidare behöver $\binom{n}{k_1,k_2,\ldots,k_m}$ förklaras. Detta kallas för *multinomialtal* eller *multinomialkoefficient*, som i sig en generalisering av binomialkoefficienten $\binom{n}{k}$. Multinomialkoefficienten kan skrivas om på detta vis.

$$\binom{n}{k_1, k_2, \ldots, k_m} = \frac{n!}{k_1! \cdot k_2! \cdot \ldots \cdot k_m!}$$



Figur 7: $(a + b) * (c + b)$ förenklas till $(a * b) + (b * c) + (b^2) + (a * c)$