UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

DV2: Algoritmer och problemlösning 7.5 p, 5DV169

Obligatorisk uppgift nr

5

Namn	Michael Andersson	
E-post	dv15man	@cs.umu.se
Cas-id	mian0057	@student.umu.se
Datum	2016-03-22	

Innehållsförteckning

1. Inledning

En automat eller en tillståndsmaskin är en abstrakt maskin som består av ett ändligt antal tillstånd. Maskinen kan endast vara vid ett tillstånd år gången och förflyttar sig mellan olika tillstånd beroende vad det är för symbol på den en inmatad sträng. En automat accepterar ett uttryck beroende i vilket tillstånd den befinner sig i när den läst klart strängen. Automaten som presenteras som i denna rapport är en Push-Down automat(PDA). Det är en automat som använder sig stack vilket möjliggör att automaten kan utföra avancerade beräkningar (Sipser, 2012).

Automaten som presenteras i denna rapport kommer vara lik en Push-Down Automat men denna är deterministisk och finit. Att den är deterministisk innebär att automaten förflyttar sig mellan tillstånden på en förutbestämd stig. Hur denna stig ser ut är helt beroende av vilka symboler som finns på den inmatade strängen. Att det är finit betyder att automaten kommer gå igenom hela strängen innan den avbryter sin körning.

Utöver datatypen PDA som presenteras i början kommer rapporten även presentera två applikationer som använder datatypen PDA för att lösa problem. Den första applikationen skapar en automat som ska kunna tolka en omvänd polsk notation. Den andra applikationen skapar en automat som kontrollerar om ett uttryck består av matchande parenteser. Varje applikation presenteras med en visuell bild av automaten och visar hur dessa applikationer testades. Rapporten avslutar med några avslutande reflektioner kring arbetet.

2. Systembeskrivning: Push down automat (PDA)

I kommande avsnitt beskrivs hur datatypen Push down automat(PDA) är konstruerad. Samt att dess gränsyta kommer presenteras och riktlinjer hur datatypen ska användas. Avsnittet avslutas med att beskriva vilka filer som datatypen är beroende av.

Denna push down automaten(PDA) består av en flexibel struktur som möjliggör för användaren att konstruera en egen automat med unika tillstånd och övergångar. I Tabell 2.1 visas gränsytan för automaten. Denna implementation använder sig av givnas datatyperna riktad lista och stack (Janlert & Wiberg, 2000). Datatypens uppbyggnad illustreras i Figur 2.1.

Figur 2.1 visar uppbyggnaden av den datatypen PDA. Den består av en mängd tillstånd lagrat i en lista där varje tillstånd i sig innehåller en lista med giltiga övergångar.

Datatypen PDA består av en mängd tillstånd där varje tillstånd innehåller en lista av möjliga övergångar och en variabel som indikerar om tillståndet är ett accepterande tillstånd. Varje övergång behöver i sin tur ha en kontrollfunktion för den inmatade strängen samt en för stacken. Ifall kontrollerna går igenom utförs övergångsfunktionen och förflyttar automaten till destinationstillståndet. Datatypen PDA innehåller variabeler som har koll på vilket start tillståndet är samt automatens nuvarande tillstånd. Den innehåller även en index räknare för den inmatade strängen.

2.2 Beskrivning av Gränsytan

Under detta avsnitt beskrivs gränsytan till datatypen PDA som finns presenterad i Tabell 2.1.

Användaren måste själv välja vilka tillstånd som automaten önskas ha och detta görs med operationen pda_setState(). Användaren måste lägga till möjliga övergångar för ett tillstånd genom att anropa pda_setTransitions(). Till denna operation behöver användarens tillgodose funktioner som kontrollerar villkoret för den inmatade strängen och stacken samt funktionspekare till en övergångsfunktion. Användaren behöver även meddela till vilket tillstånd

automaten ska förflytta sig till. Värt att notera: ifall automaten inte hittar någon giltig övergång kommer automaten inte acceptera strängen. Vilket användaren behöver ha i åtanke när den konstruerar en automat.

När alla möjliga tillstånd och övergångar har lagts i automaten kan användaren anropa funktionen pda_run(). Denna operation kommer köra igenom automaten med en önskad sträng. Värt att notera att datatypen PDA kör igenom hela den inmatade strängen inklusive NULL-tecknet. Automaten kommer traversera igenom den inmatade strängen och vid varje tillstånd undersökts om någon dess övergångar överensstämmer med det aktuella tecknet på strängen och värde på stacken. Hittas en giltig övergång utförs den givna övergångsfunktionen och automaten förflyttas till nästa tillstånd.

Tabell 2.1 visar gränsytan för den abstrakta dataypen PDA samt en beskrivning för varje operation.

Funktionsnamn	**Funktionsbeskrivning**
pda_empty(void)	Konstruerar en tom push down automat
pda_setState(pda, state, bool)	Skapar ett tillstånd i automaten. Har som indata tillståndets namn samt om det är accepterande eller ej.
pda_setInitialState(pda, state)	Berättar start tillståndet för automaten.
pda_setTransitions(pda, state, input_test, stack_test, transition_function, dest_state)	Lägger till en övergång mellan två tillstånd. Indata är en funktions pekare för kontroll av input strängen och för stacken. Samt en funktionspekare till övergångsfunktionen.
pda__stateLookup(pda, state)	Söker efter ett önskat tillstånd i automaten. Denna funktion bör endast användas av interna funktioner i automaten.
pda_run(pda, input_string)	Denna funktion kör igenom automaten med önskar input sträng.
pda_isStateAccepted(pda)	Denna funktion används efter **pda_run** för att undersöka ifall automaten godkände input strängen.
pda_getTopStackValue(pda)	Denna funktion hämtar översta värdet på stacken.
pda_free(pda)	Av allokerar allt minne associerat med automaten.

2.3 Användarhandledning för PDA

För att använda datatypen PDA behövs filerna som nämns i Tabell 2.2.

Tabell 2.2 visar vilka filer som datatypen PDA är beroende av samt kort beskrivning av filernas syfte.

pda.c/h	Filer för datatypen PDA.
dlist.c/h	Filer för riktad lista som används av datatypen PDA.
stack_1cell.c/h	Filer för en stack som används av datatypen PDA.

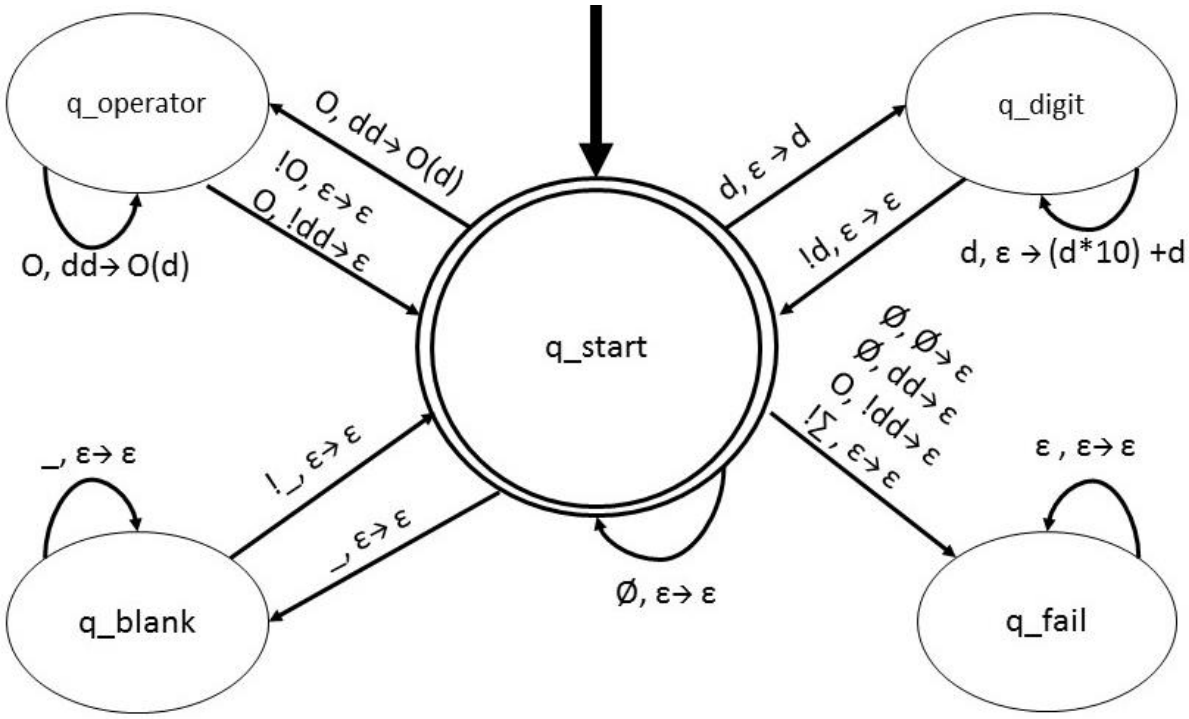
3. Applikationsbeskrivning: Omvänd polsk notation

Kommande avsnitt presenterar en applikation som använder sig datatypen PDA som presenterades i avsnitt 2. Avsnittet börjar med att beskriva automatens olika beståndsdelar och en visuell beskrivning av automatens uppbyggnad och övergångar. Avsnittet avslutas med att beskriva vilka metoder som användes för attt testa automatens korrekthet.

I Tabell 3.1 visas automaten som användes för att tolka en omvänd polsk notation. Automaten använder sig av totalt av fem tillstånd. Varje tillstånd hanterar en viss typ av tecken på input strängen vilket visas tydligare på Figur 3.1 där automaten med dess övergångar visas. Denna automat ska läsa av och beräkna positiva heltal med vanliga räkneoperationer som förekommer på den inmatade strängen. Denna automat ska även kunna hantera mellanslag. Automaten kommer placera talen från strängen på stacken som den senare kommer använda för att utföra operationerna med.

Tabell 3.1 visar den formella definitionen av automaten som tolkar en omvänd polsk notation. Q är mängden tillstånd. Σ är alfabetet automaten kan tolka. Γ är de tillåtna tecken/värden på stacken. δ är övergångsfunktioner. q_0 är start tillståndet. F är mängden accepterade tillstånd.

Q = { q_start, q_operator, q_digit, q_blank, q_fail}
Σ = {positiva heltal, + , -, /, *, mellanslag}
Γ = { heltal }
δ = Se Figur 3.1.
q_0 = q_start
F = { q_start}



Figur 3.1 illustrerar en automat som kontrollerar att ett givet uttryck är skrivit i korrekt omvänd polsk notation. Automaten består av 5 tillstånd, q_start som är start tillstånd och det enda accepterande tillståndet. Och sedan finns ett tillstånd som hanterar positiva heltal, räkne operationer, mellanslag samt icke-accepterande tecken. 'O' representerar en operator och '_' representerar ett mellanslag, 'd' står för siffra, står det 'dd' innebär det två siffror. Ifall ett utropstecken står framför en beteckning betyder det icke.

I Figur 3.1 visas hur automaten opererar och hanterar olika tecken som förekommer på den inmatade strängen. Denna automat har inget behov att lägga ett initial värde på stacken för att datatypen eftersom det finns inbyggda funktioner som kan tala om stacken är tom. Uppbyggnaden av denna automat är tämligen enkel, den startar från q_start och beroende vad det är för tecken på stacken förflyttar den sig till önskat tillstånd. Om det till exempel är det siffra på strängen förflyttas automaten till q_digit och lägger siffran på stacken. Automaten kommer stanna på q_digit tills något annat sorts tecken påträffas. Ifall flera siffror avläses utan mellanslag eller operator kommer värdet på stacken multipliceras med tio som sedan adderas med värdet på strängen.

När en operator dyker upp på den inmatade strängen kan endast operationen utföras då minst två värden finns på stacken för att operationen ska kunna utföras. Om detta villkor ej uppfylls kommer automaten ej att godkänna villkoret och visa detta genom att förflytta sig till q_fail tillståndet. Automaten förflyttar sig även till q_fail om ett otillåtet tecken påträffas i den inmatade strängen.

När automaten har traverserat igenom den inmatade strängen ska stacken endast ha ett värde på sig. Ifall den skulle ha fler än ett värde på stacken godkänner automaten inte den inmatade strängen och förflyttar sig till q_fail. Ifall det bara skulle finnas ett värde på stacken stannar automaten på q_start som är ett accepterande tillstånd.

3.1 Användarhandledning

För att köra denna applikation behövs specifika filer. De nämns i Tabell 3.2.

Tabell 3.2 visar vilka filer som behövs för att kompilera applikationen som tolkar uttryck skrivna i omvänd polsk notation samt kort beskrivning av filernas syfte.

rpn.c	Main fil för omvänd polsk notation
rpn_transitionFunktions.c/h	Filer som innehåller övergångsfunktioner
dlist.c/h	Filer för riktad lista som används av datatypen PDA.
stack_1cell.c/h	Filer för en stack som används av datatypen PDA.
pda.c/h	Filer för datatypen PDA.

För att kompilera programmet och dessa filer skrivs följande rad i terminalen:

gcc –Wall –std=c99 –o automat *.c

För att senare köra programmet skrivs den exekverbara filens namn följt av ett uttryck i omvänd polsk notationinom citationstecken:

./automat "5 5 +"

3.1 Testkörningar: Omvänd polsk notation

För att test köra detta program användes black-box testing(Myers, et al., 2011). Denna metod bygger på att testa många olika kombinationer av indata och kontrollera att applikationen ger ett förväntat svar. Resultatet från testkörningarna visas i Figur 3.2 där olika kombinationer av indata testades för att kontrollera att automaten gav ett förväntat svar. I Figur 3.2 visas att automaten inte godkänner utryck som har innehåller icke tillåtna tecken samt att den ej godkänner uttryck som försöker utföra en operation om det bara finns ett värde på stacken. Ifall automaten skulle göra division med noll avbryter automatens in körning.

Utöver black-box testning användes även white-box tesning som är en logik driven testmetod. Denna metod användes i samråd med Figur 3.1 för att kontrollera automaten gjorde korrekta övergångar där spårutskrifter användes för att undersöka vilka tillstånd automaten besökte under sin körning. På detta sätt kunde applikationens interna logik testas.

```
weasley:~/Kurser/DV2/0U5 - Loop> ./automat ""
Invalid expression
weasley:~/Kurser/DV2/0U5 - Loop> ./automat " "
Invalid expression
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "1"
1
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "11"
11
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "11 +"
Invalid expression
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "11 4 +"
15
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "11 4 + 5 /"
3
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "11 4 + 5 5 /"
Invalid expression
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "11 4 + 5 5 / a"
Invalid expression
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "10 2 * 5 / "
4
weasley:~/Kurser/DV2/0U5 - Loop> ./automat "10 2 * 5 - "
15
```

Figur 3.2 viar en delmängd av de tester som utfördes på automaten som tolkar uttryck skrivna i omvänd polsk notation..

3.2 Slutsats

Resultatet från testkörningarna visade att automaten fungerade som det var tänkt. Den kunde hantera tillåtna tecken och meddela ifall uttrycket var felaktigt samt meddela ett korrekt värde från en omvänd polsk notation.

4. Applikationsbeskrivning: Parantes matchning

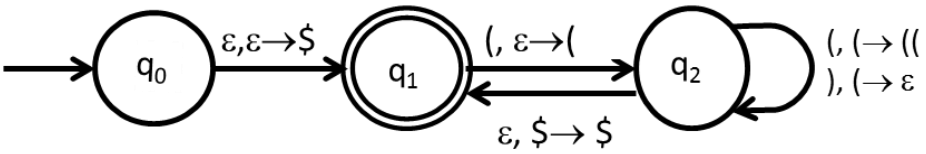
Kommande avsnitt presenterar en applikation som använder sig datatypen PDA som presenterades i avsnitt 2. Avsnittet börjar med att beskriva automatens olika beståndsdelar och en visuell beskrivning av automatens uppbyggnad och övergångar. Avsnittet avslutas med att beskriva vilka metoder som användes för attt testa automatens korrekthet.

Tabell 4.1 visar en formell definition av automaten som tolkar matchande parenteser. Den består av tre tillstånd. Denna automat tillåter endast vänster- och högerparenteser i den inmatade strängen och den behöver endast lagra vänsterparenteser på stacken för att kontrollera om det finns matchande par. Mer om detta beskrivs i nästa stycke och i Figur 4.1.

Tabell 4.1 visar den formella definitionen av automaten som tolkar matchande parenteser. Q är mängden tillstånd. Σ är alfabetet automaten kan tolka. Γ är de tillåtna tecken/värden på stacken. δ är övergångsfunktioner. q_0 är start tillståndet. F är mängden accepterade tillstånd

Q = {q_0, q_1, q_2}
Σ = { (,)}
Γ = { (}
δ = Se Figur 4.1.
q_0 = q_0
F = { q_1}

Uppbyggnaden av denna automat illustreras i Figur 4.1. När automaten körs sker en övergång från q0 till q1 som inte påverkar värdet på strängen eller stacken. Sedan kommer automaten att för varje vänsterparantes i den inmatade strängen lägga en vänsterparantes på stacken och vilket sker i q2. Detta kommer fortgå tills en högerparentes påträffas i strängen. Vid detta läge kommer en vänsterparentes avlägsnas från stacken för varje högerparentes som påträffas i strängen. Detta upprepas tills stacken är tom och då förflyttar automaten sig till q1 och godkänner tillståndet om strängen även är tom. Annars godkänner inte automaten uttrycket.



Figur 4.1 illustrerar en automat som kontrollerar om en inmatad sträng består av matchande parenteser (Westin, 2016).

4.1 Användarhandledning

För att köra denna applikation behövs specifika filer. De nämns i Tabell 4.2.

Tabell 4.2 visar vilka filer som behövs för att kompilera applikationen som testar parantesmatchning samt kort beskrivning av filernas syfte.

rpn.c	Main fil för omvänd polsk notation
brackets_transitionFunktions.c/h	Filer som innehåller övergångsfunktioner
dlist.c/h	Filer för riktad lista som används av datatypen PDA.
stack_1cell.c/h	Filer för en stack som används av datatypen PDA.
pda.c/h	Filer för datatypen PDA.

För att kompilera programmet och alla dess filer skrivs följande rad i terminalen:

gcc –Wall –std=c99 –o automat *.c

För att senare köra programmet skrivs den exekverbara filens namn följt av en mängd parenteser inom citationstecken:

./automat "(())"

4.1 Testkörningar: Parantes matchning

För att testa programmet användes främst en metod som kallas *black-box testing* (Myers, et al., 2011). Denna metod bygger på att testa många olika kombinationer av indata och kontrollera att applikationen ger ett förväntat svar. En delmängd av dessa testkörningar visas i Figur 4.2 där indata bestod av olika kombinationer av indata. Dessa testkörningar visar att applikationen godkänner en tom sträng och inte godkänner något mellanrum i strängen.

Men utöver black-box testning användes även relativt uttömmande white-box testning som är en form av logik driven test metod. I denna metod undersöktes programmets interna struktur noggrant och undersökte att övergångsfunktionerna fungerade som planerat.

```
for:~/Kurser/DV2/OU5 - Extra uppgift> ./automat ""
Accepted
for:~/Kurser/DV2/OU5 - Extra uppgift> ./automat "("
Invalid expression
for:~/Kurser/DV2/OU5 - Extra uppgift> ./automat "(("
Invalid expression
for:~/Kurser/DV2/OU5 - Extra uppgift> ./automat "(()"
Invalid expression
for:~/Kurser/DV2/OU5 - Extra uppgift> ./automat "(())"
Accepted
for:~/Kurser/DV2/OU5 - Extra uppgift> ./automat "()"
Accepted
for:~/Kurser/DV2/OU5 - Extra uppgift> ./automat "))"
Invalid expression
for:~/Kurser/DV2/OU5 - Extra uppgift> ./automat " "
Invalid expression
```

Figur 4.2 viar en delmängd av de tester som utfördes på automaten som testar parantesmatchnig.

4.2 Slutsats

Automaten fungerar som den var tänkt. Den kan hantera olika antal matchande parenteser samt meddela ifall något något otillåtet tecken förekom på strängen.

5. Avslutande reflektioner

Spännande och intressant uppgift som lärde en behovet att ha en bra struktur och plan innan man börjar med kodningsprocessen. Uppgiften var svår till en början innan man visste hur man skulle strukturera upp arbetet och hur man skulle lagra alla tillstånd och alla övergångar. Man famlade i mörkret de första dagarna men sedan lossnade det. Denna process fann jag otroligt lärorikt där man tvingades lösa ett stort problem på egen hand. När man väl hade skapat en datatypen PDA blev uppgiften tämligen enkel och man löste de små problem som dök upp under arbetets gång.

Inga övriga synpunkter på uppgiften.

Referens lista

Janlert, L.-E. & Wiberg, T., 2000. *Datatyper och Algoritmer.* 2:6 red. Lund: Studentlitteratur AB.

Myers, G. J., Badgett, T. & Sandler, C., 2011. *The Art of Software Testing.* 3:e red. u.o.:John Wiley Sons Inc.

Sipser, M., 2012. *Introduction to the Theory of Computation.* 3:e red. u.o.:South-Western College Publishing.

Westin, L. K., 2016. *Cambro - DV2: Algoritmer och problemlösning (DV169VT16): OU5 - Automat.* [Online]
Available at: https://www.cambro.umu.se/portal/site/DV169VT16-1/page/afb3f0ce-f8bf-4d61-b3ca-15f1aba2e072
[Använd 21 03 2016].