RoboLinguistics

Ett textförståelseprogram

Henrik Palmér, d01hp

Inledning

För att styra robotar räcker det inte att som i science-fictionfilmer bara säga till dem vad de ska göra. De största kraftansträngningarna inom AI för industrirobotar har inte lagts på att utveckla ett begripligt användargränssnitt gentemot dem, utan snarare på deras funktion. Det har gjort att vi nu sitter här med flexibla robotar som fungerar mycket bra så länge de ska utföra sina förprogrammerade arbetsmoment, men som ofta kräver en tekniker för att programmeras om.

RoboLinguistics är ett försök att komma förbi detta. Tanken med programmet är att man ska kunna instruera roboten på naturlig engelska. För att göra detta möjligt måste programmet inte bara kunna grammatiskt strukturera meningar med olika uppbyggnad, utan även kunna lösa koreferens och slutligen förstå vad olika kommandon betyder rent praktiskt.

Programmoment

Programmet är uppdelat i olika moment som är ansvariga för olika delar av textbehandlingen. Först skapar Charniaks parser ett träd av satsdelar som beskriver meningen grammatiskt. Detta är nödvändigt för att det andra delmomentet ska fungera. Den semantiska modulen söker upp agenter, predikat och deras argument.

Slutligen bearbetar den del av RoboLinguistics som jag har skrivit denna lista och finner verb roboten är programmerad att känna igen. För verbgrupper där dessa ingår går programmet igenom alla argument och jämför dem med dem som tidigare nämnts för att förstå vilka de är.

Den semantiska modulen

Det den semantiska modulen finner de olika predikaten i utdatan från charniakparsern. För varje predikat söker den sedan efter agenten och argumenten. När den funnit argumenten måste den dessutom klassificera dem, så att agenter och direkta objekt får rätt etiketter. Till sin hjälp har den PropBank, ett corpus på många tusen ord, där olika betydelser av verb och deras argument finns utmärkta.

Innan denna modul kan användas måste den instrueras vilka predikat den ska söka efter. Därför kommer okända verb att helt ignoreras av RoboLinguistics.

Utdatan från den semantiska modulen är ett meningsträd. Detta innehåller element med en klassificering, pekare till var i texten de förekommer samt signaturträd i de fall de är argument till verb.

Detta program hanterar två sorters signaturträd: substantivgrupper och prepositioner. En substantivgrupp har som rot huvudordet i gruppen och dess satsdel: *the red ball* kommer till exempel att ha *NP/ball/NN* som rot. Denna beteckning innebär att signaturträdet är en substantivgrupp (NP) och *ball* är ett substantiv i singular (NN).

Ett signaturträd kan också vara en samling av flera signaturträd. Detta sker vid konjunktioner som *the red ball and the blue ball* och relationer som *the book on the shelf*. I det senare fallet kommer det andra underträdet vara av typen preposition. Dessa har som underträd signaturträdet för substantivet, i fallet från den föregående meningen *the shelf* . Som rot kommer denna att ha *PP/on/IN*.

Textförståelsedelen

Trots att texten nu delats upp i sina verbgrupper och argumenten till verben till stor del har identifierats är datan ännu inte redo för roboten. För det första är datan i utdatastrukturen i princip bara text, och för det andra är ingen koreferens ännu löst. Utdata från textförståelsedelen är en lista av kommandon, som *put*, *drill* och *remove,* som alla har varierande antal argument och en lista på alla *entiteter* som nämnts i texten.

Det första programmet gör är att gå igenom listan som den semantiska modulen skapat. Denna

kommer normalt att bestå av en rad argument följd av ett verb.

Argument

Argument som RoboLinguistics förstår är de som kallas *An* i PropBank (där *n* är ett nummer). När dessa dyker upp läggs deras signaturträd i en hashtabell med argumentnamnet som nyckel. Denna hashtabell är unik för varje verb.

Verb

Textförståelsedelen representerar all kommandon den känner till med klasser. Till exempel representeras verben *get* och *take* av klassen *Get*. Innan ett kommando kan användas av programmet måste systemet få reda på att det existerar. Detta görs genom att lägga till en referens till deras klass tillsammans med namnet på verbet i en hashtabell. Varje klass kan därför kopplas till flera verb.

RoboLinguistics söker upp vilken klass som är kopplad till det aktuella verbet. Om ingen hittas genereras ett felmeddelande och all vidare bearbetning av detta verb och dess argument avbryts, annars skickas namnet på verbet, hashtabellen med argument och listan på tidigare nämnda entiteter till konstruktorn med hjälp av Java-reflektion.

Vad som sedan händer beror på vilket kommando det handlar om. De mest komplicerade som är med i RoboLinguistics för tillfället är de som tar två argument: direkta objekt och platser, som är prepositioner följda av substantivgrupper. Kommandot *remove* tillsammans med några andra gör detta.

Först tar kontruktorn reda på signaturträderna för det direkt objektet och för platsen. Om den finner båda dessa innebär det att den semantiska modulen fungerat felfritt. Detta sker dock sällan för mer komplicerade substantivgrupper med både adjektiv och prepositioner, för vilka både objekt och plats hamnar som samma argument. I dessa fall försöker kontruktorn bryta upp objektargumentet i en del före prepositionen och en med prepositionen och dess substantivgrupp.

Konstruktorn försöker sedan lösa koreferenser för att komma fram till vilka entiteter som ska läggas i dess argumentlista. För argument som kan vara olika prepositioner, som till exempel *put on* och *put into* skapas en instans av en underklass till *Preposition*, som innehåller en referens till en entitet och en specifikation, men om prepositionen är entydig läggs bara entiteten till för enkelhets skull. Vad som görs bestäms entydigt av kommandot.

Koreferens

Att lösa koreferenser är RoboLinguistics största utmaning. Klassen som har hand om detta är *EntityContainer*. Denna upprätthåller en lista över alla entiteter som någonsin nämnts. Entiteter representeras av klassen *Entity* med attribut för med vilken signaturträd den introducerades, vid vilka andra signaturträder den benämnts och som vilket argumentnummer den senast förekom, eller dess underklasser *RelationalEntity*, som är substantivgrupper som *the lid of the box* eller *the book on the table* med attribut för vilken entitet den är relaterad till och hur, och *MultipleEntities*, som är uppräkningar av flera substantiv, som *the small gerbil and the brown hamster*.

Om alla substantiv i uppräkningen har samma huvudord sätts signaturträden för entiteten till plural för detta. Vad plural av detta ord är bestäms genom att det först jämförs med ord som helt förändras, som *index*/*indice*. Om ordet inte finns i denna lista jämförs det med ord som får en förändrad ändelse, som *cactus*/*cactii* eller *box*/*boxes*. Om det inte återfinns här heller läggs helt enkelt ett s till slutet.

För att komma runt problemet att den semantiska modulen inte alltid lyckas hitta platsargument har en förenkling gjorts i *MultipleEntities*; om en uppräkning av entiteter slutar med en *RelationalEntity* antas alla entiteter i uppräkningen vara av samma klass och med samma relation. Detta innebär att *the lid and the contents of the box* förstår som *the lid of the box and the bottom of the box*, men också att *the lid of the box and book on the table* förstås som *the lid on the table and the book on the table*.

EntityContainer har en metod *findEntity()* som tar som parameter en signaturträd och vilket argument till ett kommando denna har. Därefter skapas en lista av entiteter baserad på denna. Om signaturträden inte är en konjunktion av substantivgrupper kommer listan bara att innehålla ett element.

Dessa element jämförs sedan med de redan existerande entiteterna. Om signaturträden för entiteten är ett pronomen jämförs först numerus för de olika entiteterna. Om dessa stämmer ges en grundpoäng som sedan ökas beroende på om den existerande entiteten någon gång tidigare benämnts vid detta pronomen och i så fall hur länge sedan. Detta behövs för meningar som den i *Exempel 1*, där det sista kommandot annars skulle ha blivit *glue the screw to the small box*.

När denna poäng slutligen räknats ut divideras den med ett tal i storleksordningen av det argumentnummer den existerande entiteten tidigare varit. Detta görs eftersom programmet antar att ett pronomen oftare refererar tillbaka till ett direkt objekt än platsargument. Utan detta hade det andra kommando i *Exempel 1* blivit *put a screw in the big box*.

Om signaturträden istället är en substantivgrupp jämförs först dess huvudord med dem för de redan kända entiteterna. Om dessa stämmer jämförs varje ord av typen substantiv och adjektiv i den kortaste signaturträden med alla ord i den längre. För varje matchande ord ökas den totala poängen, medan den minskas för varje ord som inte återfinns i den längre listan. Därefter delas poängen med antalet jämförda ord.

Om båda entiteterna är relationsentiteter jämförs även dessa, och utfallet av denna jämförelse får större vikt än den tidigare, eftersom programmet måste se skillnad på *the big red lid of the box* och *the big red lid of the coffin*. Slutligen delas poängen med ett tal i storleksordningen roten av avståndet i entiteter mellan den nyskapade entiteten och den tidigare kända.

Om ingen tidigare entitets poäng överstiger ett tröskelvärde behålls den nyskapade entiteten och läggs först i listan på kända entiteter, annars läggs istället den matchande entiteten först. På så sätt ligger alltid den senast nämnda entiteten först.

I de fall då den semantiska modulen inte funnit ett platsargument och istället slagit ihop det med objektargumentet uppstår ett problem; istället för att skapa en entitet och ett prepositionsobjekt skapas istället en relationsentitet, och roboten kommer att tro att den ska *ta ”boken från hyllan”* och inte *ta ”boken” från hyllan*. För att råda bot på detta finns möjligheten att bryta upp en relationsentitet till en enkel entitet och en signaturträd som beskriver relationen. Detta görs i kontruktorerna till vissa kommandon om vissa villkor uppfylls.

För det första måste relationen vara den senast introducerade entiteten. Om *the book on the* shelf redan nämnts kan det inte handla om att lägga boken på hyllan. För det andra måste relationen vara av en typ som är vettig för kommandot. För kommandot *open* är till exempel relationen *the door with the key* vettig.

Framtida utvidgningar

Det stora felet med koreferenslösningen är att konjunktionera av relationer begränsas till att vara relationer till samma entitet, och detta måste åtgärdas i kommande versioner av programmet.

Jag har medvetet valt att göra programmet så lite beroende av de underliggande modulerna som möjligt. Det innebär att förutom att dela upp relationer försöker det inte att korrigera misstag från charniakparsern eller den semantiska modulen. Det är härifrån i princip alla felaktiga tolkningar kommer. Till exempel lyckas programmet inte se sambandet mellan *the prickly cactus and the tall cactus* och *the cactii* eftersom charniakparsern tror att *the cactii* är singular.

Programmet antar att varje verb som förekommer i texten är en uppmaning. Detta anser jag vara ett rimligt antagande, eftersom poängen med det är just att finna kommandon i en löpande text. Alternativet vore att ignorera verb som inte är uppmaningar, men det skulle leda till att information från texten gick förlorad. Detta leder dock till fel som i *Exempel 3*.

De fel som stammar från det sista steget beror till största delen på att entiteterna bara minns hur de benämnts och vilka roller de spelat och inte till vilka verb de spelat dessa roller. Detta illustreras av *Exempel 4*. För att lösa detta måste den modularitet där det är lätt att lägga till nya kommandon

ge vika för ett system där kommandon känner till varandra och vet hur de ska förhålla sig till varandras argument. I denna övervägning prioriterar jag enkelhet.

Något som däremot måste läggas till om programmet ska få någon nytta är en databas med objekt tillgängliga för roboten. I nuläget representeras kommandon av klasser som är kopplade till en handling, medan entiteterna bara beskrivs av strängar.

Appendix

Exempel 1

”Remove the lid from the big box and put a screw in it, and then glue it to the small box.”

```
remove the lid (E1) from the big box (E2).
put a screw (E3) in the lid (E1).
glue the lid (E1) to the small box (E4).
--------------------------------------------------------------------------
Known entities:
the lid (E1)
  it
  the lid
the small box (E4)
  the small box
a screw (E3)
  a screw
the big box (E2)
  the big box
```

Exempel 2

”Open the big box and remove a screw from it. Put the screw in the center of the lid of the box and place the lid on the box. Please glue a handle to the lid of the small box. Open the small box and put an apple in it and then close the box.”

I detta exempel finns specificeraren *in the center* i den andra meningen. I det tredje kommandot har därför *the center* inte något ID; det existerar inte som en entitet. Den tredje meningen dyker inte upp överhuvudtaget, vilket beror på att charniakparsern felaktigt identifierat *glue* som ett substantiv: ”(S1 (S (NP (NN Glue)) (DT a) (VP (VB handle) (PP (TO to) (NP (NP (DT the) (NN lid)) (PP (IN of) (NP (DT the) (JJ small) (NN box)))))) (. .))).”

```
open the big box (E1).
remove a screw (E2) from the big box (E1).
put a screw (E2) in the center of the lid (E3) of the big box (E1).
put the lid (E3) of the big box (E1) on the big box (E1).
open the small box (E4).
put an apple (E5) in the small box (E4).
close the small box (E4).
--------------------------------------------------------------------------
Known entities:
the small box (E4)
  the box
  it
  the small box
an apple (E5)
```

```
  an apple
the lid (E3) of the big box (E1)
  the lid
the big box (E1)
  the box
  it
  the big box
a screw (E2)
  the screw
  a screw
```

Exempel 3

”He opened the door.”

```
open the door (E1).
--------------------------------------------------------------------------------
Known entities:
the door (E1)
  the door
```

Exempel 4

”Open the box with the key and then open the door with it.”

I detta exempel illustreras hur programmet förvirras av grammatiska konstruktioner som en människa lätt genomskådar. Eftersom systemet inte känner till att man vanligtvis öppnar saker med nycklar och inte lådor säger dess regler åt den att välja lådan som instrument för att öppna dörren.

```
open the box (E2) with the key (E1).
open the door (E3) with the box (E2).
--------------------------------------------------------------------------------
Known entities:
the door (E3)
  the door
the box (E2)
  it
  the box
the key (E1)
  the key
```