Øving 5 – Soar

Frist: 10/11

I denne øvingen skal vi se nærmere på Soar, både fra et teoretisk perspektiv men også på en praktisk måte. Noen av oppgavene vil ha tema fra det klassiske problemet ”wolf-cabbage-goat”:

En bonde skal frakte en ulv, en sau og et salathode over en elv. Han kan bare ro en ting av gangen over. Problemet er at hvis bonden ikke er tilstede, vil ulven spise sauen og sauen spise salathodet.

Oppgave 1-4 er obligatoriske, mens oppgave 5 er frivillig.

Spørsmål 1: The Soar processing cycle

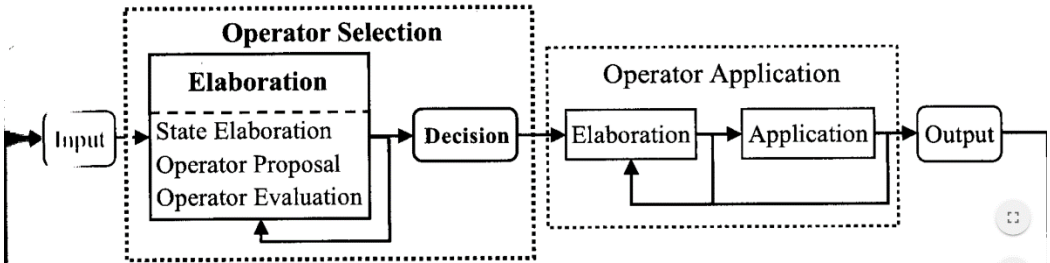


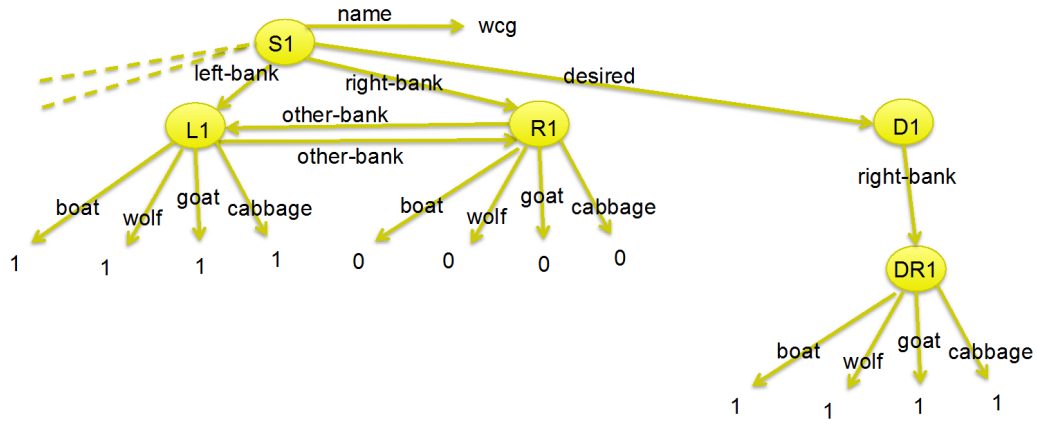
Figure 4.9
Selecting an operator.

I figur 4.9 kan du se en illustrasjon av prosesseringssykelen til Soar. Beskriv hva som skjer i de følgende fasene: Input phase, Operator Selection og Operator Application.

Oppgave 2 - diverse spørsmål

a) **Hva er forskjellen på i-support og o-support?**
b) **Hva er en “impasse” og hvordan håndterer Soar situasjonen?**

Oppgave 3 – forståelse av representasjon



I figuren over ser man tilstandsrepresentasjonen til en Soar-agent. Tilstanden beskriver starttilstanden i ”wolf-cabbage-goat”.

a) Gi eksempler på følgende fra figuren: Identifikatorer, attributter og verdier.
b) Hva slags type er DR1?

Spørsmål 4: Kodeforståelse.

Nedenfor ser du tre kodesnutter fra et Soar program som er med å løse "Wolf-Cabbage-Goat"-problemet. For hver kodesnutt skal du avgjøre hva slags type regel det er, for eksempel om det er en evalueringsregel. Forklar hvordan du tenker.

```
sp {wcg*------*-------*move-boat-cargo
   (state <s> ^name wcg
              ^<< right-bank left-bank >> <bank>)
   (<bank> ^{ << wolf goat cabbage >> <type> } > 0
           ^boat 1)
-->
   (<s> ^operator <o> + =)
   (<o> ^name move-boat
        ^bank <bank>
        ^cargo <cargo> )
   (<cargo> ^<type> 1)
}
```

```
sp {wcg*------*-------*move-boat*record*last-operator*alone
   (state <s> ^name wcg
              ^operator <o>)
   (<o> ^name move-boat
        ^bank <bank>
        ^cargo <cargo>)
   (<cargo> ^empty 1)
-->
   (write (crlf) | record:| alone |  |)
   (<s> ^last-operator <o1>)
   (<ol> ^bank <bank>
         ^cargo <ocargo> )
   (<ocargo> ^empty 1)}
```

```
sp {wcg*------*-------*------*inverse*failure
   (state <s> ^name wcg
              ^operator <o> +
              ^failure <d>
              ^last-operator <lo>)
   (<o> ^name move-boat
        ^cargo <cargo>)
   (<cargo> ^<type> <number> )
   (<lo> ^cargo <ocargo>)
   (<ocargo> ^cargo <ocargo>)
   (<ocargo> ^<type> <number>)
-->
   (write (crlf) | -----------| <type> |  |
   <number> |    |)
   (<s> ^operator <o> >) }
```

Oppgave 5 (frivillig)

I denne oppgaven skal du skrive noen regler i Soar for å løse Ulv-,Geit- og Salat-problemet nedenfor. Som startpunkt får du utdelt en god del av koden, og din oppgave er å programmere de delene av Soar-produksjonsreglene (**SP**) som mangler.

Øvingen forutsetter at du har installert SoarSuite9 fra https://soar.eecs.umich.edu/Downloads

Koden ligger i mappen **oving5H17** i vedlegget, og inneholder prosjektfilen **wcg.vsa** som du må åpne i VisualSoar. Du finner deretter SPene du skal skrive i VisualSoar-mappene (uthevet) til venstre i vinduet. Følgende SPer skal her programmeres (mappen/filen hvor de ligger er utvhevet):

intialize-mac :
wcg*propose*initialize, wcg*apply*initialize

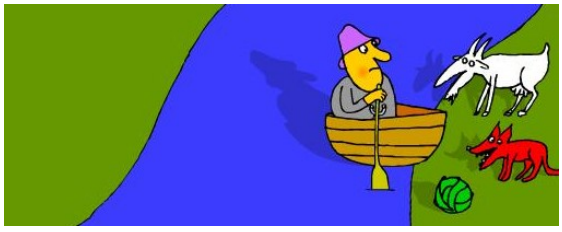
elaborations/goal-test:
wcg*detect*state*success

move-boat:
wcg*propose*operator*move-boat-cargo,
wcg*propose*operator*move-boat-empty
apply*move-boat-cargo
apply*move-boat-alone

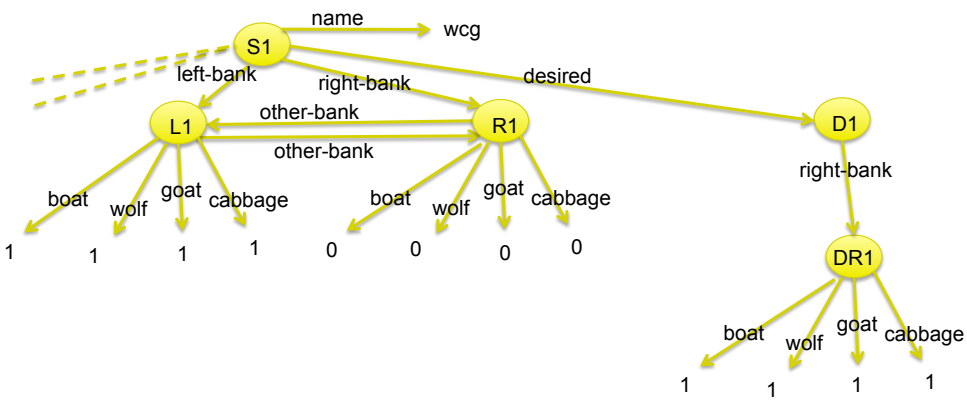
elaborations/search-control:
wcg*select*operator*prefer*inverse*failure,
wcg*select*operator*avoid*inverse*not*failure

Mer detaljert info om hvordan disse SPene skal programmeres finner du i delspørsmålene 1 - 5 litt lengre ned i oppgaven. Det kan uansett lønne seg å kikke litt rundt på de ulike kodebitene før du går i gang. For å teste og kjøre wcg, må du overføre prosjektet til SoarJavaDebugger-agenten med ”Send all Files” og deretter trykke agentens ”Run”-knapp. ”Expose” sletter koden i agenten og gjør klar for ny opplasting.

Så til problembeskrivelsen: En bonde skal frakte en ulv, en sau og et salathode over en elv. Han kan bare ro en ting av gangen over. Problemet er at hvis bonden ikke er tilstede, vil ulven spise sauen og sauen spise salathodet. Vi skal altså lage et program i Soar som løser problemet, dvs ror alle de ulike objektene over elva (random søk i et tilstandsrom).



1. Nedenfor ser du en representasjon som skal brukes oppgaven:



Attributtet `^desired` angir ønsket måltilstand.

Initialiserer problemet med å lage strukturen i figuren med ønsket måltilstand (alle over på høyre elvebredd), dvs åpn filen **intialize-mac** og skriv koden for wcg*propose*initialize og wcg*apply*initialize.

Merk også at (`<s> ^name wcg)` angir navnet på problemrommet og forteller at det er initialisert.

2. Åpn filen **elaborations/goal-test** og skriv wcg*detect*state*success som detekterer at måltilstanden er nådd og stopper agenten.
3. Den valgte representasjonen gjør det lett å skrive ut status på elvebreddene og bruke enkel aritmetikk til å telle opp og ned ettersom objekter flytter seg. Åpn filen **move-boat** og skriv wcg*propose*operator*move-boat-cargo, wcg*propose*operator*move-boat-empty, apply*move-boat-cargo og
apply*move-boat-alone.

   La operatorene ha indifferent preferanse (Soar velger da blant operatorene random). Dette kan gjøres ved: `(<s> ^operator <o> + =)` eller alternativt i to steg ved: `(<s> ^operator <o> +)(<s> ^operator <o> =)`.

Operatorene har attributter `^name`, `^bank`, og `^cargo`. Lasten `^wolf`, `^goat`, og `^cabbage` legges under attributtet `^cargo` for å lette matching senere. La `(<cargo> ^empty 1)` bety at bonden ikke tar med seg noe last, dvs ror alene over.

4. Hvis siste overfart ledet til en **feil** (geit-salat alene etc), skal dette reverseres. Vi bruke attributtet `^failure` for å indikere feil. Den inverse aksjonen gjøres ved å utføre siste operator en gang til (ror da tilbake med evt. last).
   Inverteringen skjer ved å gi siste operator høy preferanse (best ”>”). Åpn filen **search-control** i mappen **elaborations**. Her skal du skrive
   wcg*select*operator*prefer*inverse*failure som sammenligner foreslåtte operatorer mot operatorbeskrivelsen under `^last-operator`, og setter ”best”-preferanse på den operatoren som er lik (matcher). Ferdig kode, som tar vare på siste operator, ligger i filen **move-boat.** SPer som detekterer feiltilstand og lager `^failure`, ligger i filen **goal-test** i mappen **elaborations.**

5. Hvis alt er OK etter en overfart , dvs –`^failure` (not failure), vil vi unngå å ro tilbake med lasten. Dette kan gjøres ved å gi siste operator lavere preferanse (worst ”<”). Skriv wcg*select*operator*avoid*inverse*not*failure som gjør dette.

Dersom du har gjort 1 - 5 skal du nå kunne la Soar løse problemet ved å kjøre Soar-agenten i SoarJavaDebugger ☺