729G78 Artificiell intelligens Kunskapsrepresentation – Predikatlogik 2

Arne Jönsson

HCS/IDA

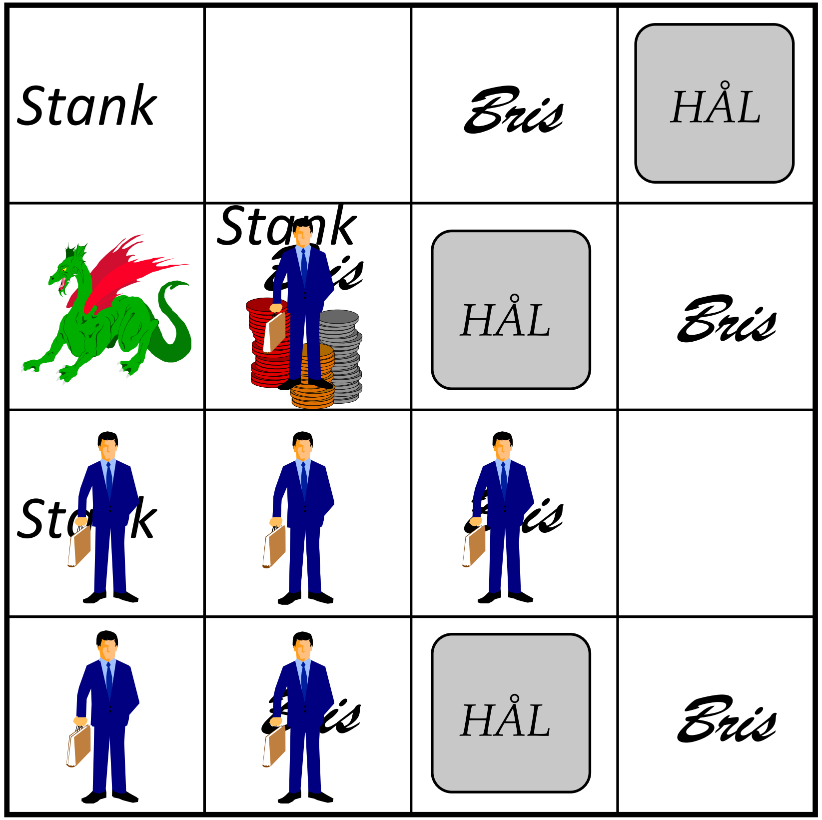
Kunskapsbaser

Skapa en kunskapsbas

Iterativ process

1. Identifiera uppgiften
2. Hämta relevant domänkunskap
3. Definiera en vokabulär
   - Ontologi
4. Skriv axiom som beskriver domänen
5. Gör en problemkodning
6. Testa och använd
7. Debug

Wumpus-världen



Wumpus-världen

- `Percept = [Stench, Breeze, Glitter, Bump, Scream]`
- `Action = [Forward, Right, Left, Grab, Shoot, Climb]`
- Representera tid och välja handling
- Ex
  - Antag `Percept([s, b, Glitter, m, c], 5)`
  - `ask(∃a bestAction(a, 5))` bör ge `Grab`
  - `tell(Grab, 5)`

Representation av världen

- Rutor [x, y]
- Närliggande rutor
  - $\forall x, y, z, w \; Adjacent([x, y],[z, w]) \Leftrightarrow [z, w] \in \{[x+1,y], [x-1,y], [x, y+1], [x, y-1]\}$
- Lagra utforskning, ex
  - $\forall s,t \; At(Agent, s, t) \land Breeze(t) \Rightarrow Breezy(s)$
- Diagnosregler (diagnosticerar från effekter till fakta)
  - $\forall s \; Breezy(s) \Rightarrow \exists r \; Adjacent(r, s) \land Pit(r)$
- Kausalregler (modellbaserade resonemang)
  - $\forall r \; Pit(r) \Rightarrow (\forall s \; Adjacent(r, s) \Rightarrow Breezy(s))$

Hantera föränding

- Vill uttrycka hur en handling påverkar världen i FOPL
- Tidsvariabeln används för sådant som ändrar sig
  - Ex At(Agent, [1, 1], T_0), At(Agent, [1, 2], T_1)
  - Saker som inte ändrar sig behöver ingen tidsvariabel, ex Wall([0, 1])
  - Uppdatera världen
  - Action(Forward, T_1) $\longrightarrow$ At(Agent, [1,3], T_2)
  - Action(Grab, T_2) $\longrightarrow$ Holding(Gold, T_3)

Hantera förändring

- Världen består av en sekvens **situationer**
- **Handlingar** resulterar i nya situationer
- **Fluents** är predikat som beskriver relationer eller egenskaper som kan påverkas av handlingar

Exempel

- Beskrivning av wumpusvärldens initialtillstånd
  - At(Agent, [1 1], T_0)
  - Direction(Agent, North, T_0)
  - At(Wumpus, [1 3])
  - At(Pit1, [3 1])
  - At(Pit2, [3 3])
  - At(Pit3, [4 4])
  - At(Gold, [2 3])

Effektaxiom

- Wumpusvärldens handlingar
  - $\forall$x,y,t At(Agent, [x, y], t) $\wedge$ Direction(Agent, North, t) $\wedge$ Action(Forward, t) $\Rightarrow$
    At(Agent, [x, y+1], t+1)
  - $\forall$a,t Direction(Agent, East, t) $\wedge$ Action(Left, t) $\Rightarrow$
    Direction(Agent, North, t+1)
  - $\forall$x,y,t At(Agent, [x, y], t) $\wedge$ At(Gold, [x, y], t) $\wedge$ Action(Grab, t) $\Rightarrow$
    Holding(Agent, Gold, t+1)
  - Kallas effekt-axiom, beskriver det som förändras

Förändringar i världen

- Antag
  At(Agent, [2,3], T_0), At(Gold, [2, 3], T_0)
  Grab ger Holding(Gold, T_1)
- Antag att agenten går framåt
  At(Agent, [2, 3], T_1) $\wedge$ Direction(Agent, North, T_1) $\wedge$ Action(Forward,T_1) $\Rightarrow$
  At(Agent, [2, 4], T_2)
- Men agenten har också guldet med sig, dvs
  $\forall$ x,t Holding(x, t) $\wedge$ $\neg$ Action(Release, t) $\Rightarrow$ Holding(x, t+1)
- och
  $\forall$ x,t $\neg$Holding(x, t) $\wedge$ $\neg$ Action(Grab, t) $\Rightarrow$ $\neg$Holding(x, t+1)
- Kallas frameaxiom

Frameproblemet

- Hur representera vad som påverkas och vad som inte påverkas av en handling.
  - Representationsproblemet: representation av handlingar
  - Inferensproblemet: representation av sekvenser av handlingar
- Relaterade problem:
  - Qualification problem
    - Hur räknar vi upp under vilka omständigheter en handling lyckas
  - Ramification problem
    - Hur räknar vi upp alla implicita konsekvenser av en handling

Succesor-state-axiom

- **Successor-state axiom** beskriver hur fluents förändras över tid
  - Kombinera effekt-och frameaxiom
  - Handling som gör något sant
    ∨
    redan sant och ingen handling som gör falskt
- $\forall$x,y,t At(Agent, [x+1, y], t+1) $\Leftrightarrow$
  (Action(Forward,t) $\wedge$ At(Agent, [x, y], t) $\wedge$ Direction(Agent, East, t))
  $\vee$
  ($\neg$ Action(Forward,t) $\wedge$ At(Agent, [x+1, y], t))

Inferens för predikatlogik
Unifiering

LINKÖPINGS UNIVERSITET

Resolution för predikatlogik

Variabler och kvantifierare

Ex

1. At(Wumpus, [1, 2], 5)
2. ¬At(Wumpus, [1, 3], 6)

Ger resolventen ¬At(Wumpus, [1, 3], 6) ∨ At(Wumpus, [1, 2], 5)

medan

3. ¬At(Wumpus, l, t) och 1.

med {l/[1, 2], t/5} (l substitueras med [1, 2] och t substitueras med 5)

ger ¬At(Wumpus, [1, 2], 5) och en motsägelse, dvs en tom klausul

Unifiering

- För att lyckas med resolution i FOPL behöver man hitta alla lämpliga substitutioner, kallas unifiering
- Variabler byts mot andra variabler eller konstanter, en substitution
- Ofta söker man den mest generella unifieringen, MGU

Unifiering – Exempel

Gillar(Olga, x) Gillar(y, mask)

x/mask y/Olga

Gillar(Olga, mask)

Gillar(Olga, x) och Gillar(y, mask) kan alltså unifieras med hjälp av substitutionen {x/mask, y/Olga)

Substitutioner propageras

Ex

$\forall x$ Hund(x) ⇒ Skäller(x) kan konverteras till

1. ¬Hund(x) ∨ Skäller(x) om vi antar att variabler alltid är allkvantifierade

2. Hund(Pluto)

1+2 med [x/Pluto] ger Skäller(Pluto), dvs x har substituerats i båda predikaten

Unifieringsalgoritmen, 1

```
def unify(x, y, subst):
    if subst=="Fail":
        return "Fail"
    elif x==y:
        return subst
    elif variable(x):
        unifyVar(x, y, subst)
    elif variable(y):
        unifyVar(y, x, subst)
    elif isinstance(x, list) and isinstance(y, list):
        unify(rest(x), rest(y), unify(first(x), first(y), subst))
    else:
        return "Fail"
```

Unifieringsalgoritmen, 2

```
def unifyVar(var, x, subst):
    if getSubst(var, subst):
        unify(lookup(var, subst), x, subst)
    elif variable(x) and getSubst(x, subst):
        unify(var, lookup(x, subst), subst)
    elif occursIn(var, x, subst):
        return "Fail"
    else:
        extendSubst(var, x, subst)
```

Exempel

Unifiera `Smelly([1,2])` och `Smelly(k)`

`unify([”Smelly”,”[1,2]”], [”Smelly”,k],[])`

`x = [”Smelly”,”[1,2]”], y = [”Smelly”,k]`

1. `subst≠Fail`
2. `x≠y`
3. `x` eller `y` inte variabler
4. `x` och `y` listor
5. `unify(”[1,2]”, k, unify([”Smelly”], [”Smelly”],[]))`
6. `x==y` alltså inga nya substitutioner
7. `k` variabel så `unify` ger anrop till `unifyVar(k, ”[1,2]”,[])`
8. `unifyVar` hittar inga substitutioner så
9. `extendSubst(k, ”[1,2]”, [])` returnerar `{k/”[1,2]”}`

Större exempel (med förenklingar)

unify([’ F’ ,[’ G’ , ’ A’ , m], [’ F’ , k, m]], [’ F’ , l, [’ F’ , l, [’ G’ , ’ A’ , ’ B’]]], [ ])

unify([[’ G’ , ’ A’ , m], [’ F’ , k, m]], [l, [’ F’ , l, [’ G’ , ’ A’ , ’ B’]]], unify(’ F’ , ’ F’ , [ ])) ⟶ [ ]

unify([[’ G’ , ’ A’ , m], [’ F’ , k, m]], [l, [’ F’ , l, [’ G’ , ’ A’ , ’ B’]]], [ ])

unify([’ F’ , k, m], [’ F’ , l, [’ G’ , ’ A’ , ’ B’]], unify([’ G’ , ’ A’ , m], l, [ ]))

unify-var(l, [’ G’ , ’ A’ , m], [ ]) ⟶ [l/[’G’, ’A,’ m]]

unify([k, m], [l, [’ G’ , ’ A’ , ’ B’]], unify(’ F’ , ’ F’ , ’ [l/[’G’, ’A’, m]]’)) ⟶ [l/[’G’, ’A’, m]]

unify([k, m], [l, [’ G’ , ’ A’ , ’ B’]], ’ [l/[’G’, ’A’, m]]’))

unify(m, [’ G’ , ’ A’ , ’ B’], unify(k, l, ’ [l/[’G’, ’A’, m]]’))

unify-var(k, l, [l/[’G’, ’A’, m]])

unify(k, [’ G’ , ’ A’ , m], ’ [l/[’G’, ’A’, m]]’)

unify-var(k, [’ G’ , ’ A’ , m], ’ [l/[’G’, ’A’, m]]) ⟶ [k/[’G’, ’A’, m]]

unify(m, [’ G’ , ’ A’ , ’ B’], ’ [k/[’G’, ’A’, m], l/[’G’, ’A’, m]]’)

unify-var(m, [’ G’ , ’ A’ , ’ B’], ’ [k/[’G’, ’A’, m], l/[’G’, ’A’, m]]’) ⟶ [m/[’G’, ’A’, B]]

[m/[’G’, ’A’, ’B’], k/[’G’, ’A’, m], l/[’G’, ’A’, m]]