Kunstmatige Intelligentie opdracht 2

Multi-agent systems

2 maart 2012

1 Introductie

De tweede opdracht van het vak Kunstmatige Intelligentie gaat over multi-agent systems. Hierbij wordt gebruik gemaakt van de programmeertaal 2APL (spreek uit: double APL, A Practical Agent Programming Language). Je kunt 2APL downloaden van `http://apapl.sourceforge.net`. Er wordt een uitgebreide handleiding meegeleverd (`manual.pdf`).

In Sectie 2 worden de basisprincipes van 2APL uitgelegd. De opdracht bestaat uit het aanpassen van twee bestaande agenten, dit wordt verder uitgelegd in Sectie 3. In de Appendix staat een uitgebreide beschrijving van de bestaande agenten.

2 Basisprincipes 2APL

De syntax van 2APL is gebaseerd op Prolog. Het 2APL-platform is geïmplementeerd in Java. Dit platform kan programma's met 2APL-code uitvoeren. Een multi-agent system in 2APL is gedefinieerd als een verzameling autonome agenten in een omgeving die via het platform met elkaar communiceren. De omgeving is een eigen programma dat reageert op de acties van de agenten.

De verzameling agenten en de omgeving is gedefinieerd in een xml-bestand met als extensie .mas. De syntax van dit bestand is als volgt:

```
<apaplmas>
  <environment name="" file="*.jar">
    <parameter key="" value=""/>
  </environment>
  <agent name="" file="">
   <beliefs file="" shadow="true|false"/>
   [overige beliefs]
  </agent>
  [overige agents]
</apaplmas>
```

Bij enkelvoudige tags (zoals `<parameter/>` staat de sluiting aan het eind van de tag (met `/>`), meervoudige tags worden afgesloten met `</tag>`. De environment

wordt in de huidige opdracht niet gebruikt. De agent-tags geven de naam en de broncode van de agent. De naam moet uniek zijn voor elke agent, de broncode mag hetzelfde zijn (om meerdere instanties van dezelfde agent in het systeem te plaatsen). Het element `<beliefs>` bevat een externe belief base (zie onder voor uitleg). Deze externe belief base kan worden gebruikt om eenvoudig agents met dezelfde broncode van elkaar te onderscheiden omdat ze verschillende beliefs hebben (bijvoorbeeld een verschillende startpositie). De belief-tag heeft een attribuut `shadow`. Dit attribuut bepaalt of de beliefs in de externe beliefbase zichtbaar zijn in de interface. De beliefbase wordt onder andere gebruikt om axioma's te definiëren, het is vaak handig om deze te verbergen zodat de overige beliefs overzichtelijk worden weergegeven.

De agenten werken volgens een formele logica met de concepten *Belief*, *Goal* en *Plan*. Goals zijn doelen die een agent probeert te bereiken. Plans zijn manieren om deze doelen te bereiken. Beliefs zijn de overtuigingen die de agent heeft over de staat van het systeem en de relaties tussen de concepten uit het systeem.

Een agent probeert altijd om zijn doelen te bereiken. In elke iteratie (program cycle) wordt gekeken wat de relevante plans en beliefs zijn voor de huidige doelen en wat op basis daarvan de stappen zijn die het doel bereiken of dichterbij brengen. Een doel kan bijvoorbeeld zijn: ga naar positie 3,5. Een plan kan zijn: als je op positie A,B bent en je wilt naar positie X,Y dan moet je X-A stappen in horizontale richting en Y-B stappen in verticale richting bewegen. Een belief kan zijn: huidige positie is 2,2. Beliefs, plans en goals zijn dynamisch: ze kunnen worden aangepast, verwijderd of toegevoegd door de agent (overigens niet door andere agents of door de omgeving, agenten zijn autonoom).

De code voor een 2APL-agent bestaat uit zeven onderdelen: Beliefs, Goals, Plans, Belief Updates, PG-rules (Plans specifiek voor Goals), PR-rules (Plan Repair, aanpassingsregels voor Plans) en PC-rules (procedural rules: implementatie van abstracte regels in Plans).

De interface bevat een overzicht van de zeven onderdelen van de agent. Verzonden en ontvangen berichten staan in de tab `Log`. In de knoppenbalk staan oa. de knoppen `Start deliberation`, `Perform one deliberation cycle` en `Perform one deliberation step`. De eerste twee hiervan zijn handig om te gebruiken. Met de eerste knop begint het multi-agent system te draaien. Met de tweede knop wordt 1 cycle uitgevoerd, waarbij alle onderdelen worden gecontroleerd (start een plan voor een goal, voer van huidige plannen 1 stap uit, verwerk berichten).

3 Opdracht

De opdracht gaat over een multi-agent system voor het vervoeren van pakketten. Er zijn aanbieder-agents die een pakket willen laten vervoeren naar een bepaalde lokatie en vervoerder-agents die vervoeropdrachten kunnen accepteren en uitvoeren. De vervoerders kunnen een bod uitbrengen op een bepaalde opdracht en de aanbieders beslissen aan welke agent de opdracht wordt toege-

kend. Na het uitvoeren van de opdracht worden de kosten voor de vervoerder verrekend met de afgesproken prijs, en op basis van de winst of verlies kan de vervoerder een bod op een toekomstige opdracht aanpassen.

Opgave 1 en 2 gaan over de aanbieder-agent. Opgave 3-6 gaan over de vervoerder-agent. De vervoerder-agent is complementair aan de aanbieder-agent. De vervoerder-agent moet een bod uitbrengen op aangeboden opdrachten en geaccepteerde opdrachten uitvoeren. De hoogte van toekomstige biedingen moet worden aangepast aan het resultaat van uitgevoerde opdrachten.

Opgave 1. De belief base van een agent kan worden aangepast met *Belief Update Rules*. Uitleg en voorbeelden van deze regels staan in de code en in de Appendix. Schrijf zelf een belief update-regel voor de aanbieder-agent die de lijst biedingen voor een bepaalde OID uit de belief base verwijdert.

Opgave 2. Een plan uit de plan base van een agent kan worden uitgevoerd met *Procedural rules* of PC-rules. Pas de PC-rule kiesV van de aanbieder-agent aan. Schrijf een Prologfunctie `keuze(L,Vt,Va,Pt)` die gegeven een lijst biedingen `L` (met als elementen `bod(Agent,Prijs)`) de variabelen `Vt`, `Va` en `Pt` bindt aan de agent met het beste bod, het andere bod, en de hoogte van het beste bod, respectievelijk. Stuur de vervoerder-agenten een bericht met het resultaat van de keuze. Beindig de PC-rule met het verwijderen van de lijst biedingen mbv. de belief update-regel uit opgave 1. Voor deze opgave zijn twee vervoerder-agenten nodig. Voeg in het `.mas`-definitiebestand een vervoerder-agent toe. Gebruik de bestaande agent- en belief-bestanden maar geef de agent wel een andere naam.

Opgave 3. Schrijf PC-rules voor de vervoerder-agent om te reageren op berichten van de aanbieders. Als de vervoerder al een actieve opdracht heeft is hij bezet en kan dus geen nieuwe opdrachten aannemen. Verwerk dit in een Prolog-predicaat binnen de PC-rule of maak twee aparte regels met verschillende precondities. Maak ook regels voor het verwerken van een geaccepteerd of afgewezen bod. Bij een afgewezen bod moet er een prijscorrectie-constante in de beliefbase worden aangepast.

Opgave 4. Schrijf een Prologfunctie die gegeven een nieuwe opdracht met een afzender, bestemming, aantal en de al eerder uitgevoerde opdrachten in de belief base een bod uitbrengt op de nieuwe opdracht. Houd rekening met de winst of verlies die op eerdere opdrachten is gemaakt. Bedenk een goede afstandsmaat tussen opdrachten. Een opdracht van A naar B met aantal 8 lijkt bijvoorbeeld op een opdracht van B naar A met aantal 6. Als er geen eerdere opdrachten zijn die lijken op de nieuwe opdracht kan je een willekeurig bod uitbrengen.

Opgave 5. Schrijf een PG-regel om een geaccepteerde opdracht uit te voeren. Dit plan bevat een aantal stappen: verplaatsen naar de aanbieder, inladen, verplaatsen naar de bestemming, uitladen. Bereken de winst die gemaakt is op deze opdracht. Het afleggen van 1 afstandseenheid kost 1 geldeenheid. De inkomsten zijn zoals afgesproken met de aanbieder. Als er verlies is gemaakt op een opdracht moet de prijscorrectiecosntante worden verhoogd.

Opgave 6. De belief base van de vervoerders bevat de afstanden tussen de aanbieders, in de predicaten `env_afstand(AanbiederX,AanbiederY,Afstand)`.

Neem nu aan dat de omgeving partieel observeerbaar is, en dat een afstand pas bekend is nadat de route tenminste 1 keer is afgelegd. Pas de PG-regel voor het uitvoeren van een opdracht aan zodat afgelegde afstanden worden opgeslagen in aparte predicaten `b_afstand`. Gebruik alleen de bekende `b_afstand`-predicaten bij het berekenen van een bod op een nieuwe opdracht. Afstanden zijn reflexief: `afstand(X,Y)` is gelijk aan `afstand(Y,X)`. Pas wel op voor oneindige lussen in het opvragen van de afstand.

3.1 Verslag

Het verslag moet de volgende elementen bevatten:

- Korte introductie over Belief-Plan-Goal-agents.
- Beschrijving van de architectuur van de agents in de opdracht aan de hand van de goals, plans en beliefs.
- Beschrijving van het algoritme dat wordt gebruikt om een bod uit te brengen.
- Experimenteer met verschillende aantallen aanbieder-agents (bv. 2-4) en vervoerder-agents (bv. 1-3). Maak een grafiek om de prestaties van de verschillende systemen met elkaar te vergelijken, bijvoorbeeld in termen van gemiddelde winst of winst per vervoerder-agent.

Appendix

De aanbieder-agent zal als voorbeeld worden uitgewerkt. In 2APL zijn variabelen met een hoofdletter en constantes met een kleine letter, zoals in Prolog. De aanbieder-agent heeft 1 doel: af en toe een nieuwe opdracht aanbieden. In de code staat dit in de sectie `Goals` als `geefOpdracht`. Er is een PG-rule waar een plan instaat om dit doel te bereiken. Een PG-rule heeft de volgende vorm:

```
doel <- condities |
{
plan
...
}
```

Hierbij wordt het plan gebruikt om het doel te bereiken als aan de condities wordt voldaan. De condities worden geëvalueerd mbv. de beliefs. In dit geval moet het genereren van een nieuwe opdracht gebeuren met een kleine waarschijnlijkheid, om te voorkomen dat er teveel opdrachten worden gegenereerd. De conditie is daarom `prob(0.05)`. In de belief base wordt deze conditie uitgewerkt: `prob(P) :- is(X, rand), X < P`.

De belief base is in de code van een agent aangegeven met het keyword `Beliefs`. Daarnaast kan de agent gebruikmaken van beliefs uit externe bestanden. Deze bestanden zijn in het `.mas`-definitiebestand aangegeven voor de

betreffende agenten. De agent behandelt interne en externe beliefs op dezelfde manier. Als je een belief-definitie schrijft (zoals de functie `keuze(L,Vt,Va,Pt)` uit Opdracht 2) kan je dus kiezen of je dit in het agent-bestand of in het externe bestand doet.

Het plan bestaat uit een aantal stappen: kies (willekeurig) een bestemming en een aantal, verwerk dit als opdracht in de belief base, stuur een bericht naar de vervoerder-agents met de opdracht, en voeg als nieuw doel toe het kiezen van een vervoerder voor deze opdracht. De stappen van een plan worden gescheiden met een puntkomma (`;`). Dit is een scheidingsteken, de laatste stap uit een plan wordt niet afgesloten met `;` Dit geldt voor alle code tussen accolades, ook bijvoorbeeld `if-then-else`.

Elke nieuwe opdracht moet een opdracht-id krijgen die 1 hoger is dan de vorige opdracht-id. De huidige opdracht-id staat in de belief base als `oid(0)`. In het plan voor een nieuwe opdracht wordt dit belief aangepast aan de nieuwe situatie. Dit gebeurt met de Belief Update-regel `VerhoogOID` (belief update-regels zijn met een hoofdletter). Een belief update-regel heeft een preconditie en een postconditie. Deze condities worden gevalueerd mbv. de belief base. De preconditie is in dit geval `oid(N)`, dit betekent dat in de huidige belief base wordt gezocht naar een statement dat deze conditie waarmaakt. In dit geval staat `oid(0)` in de belief base, dus de variabele `N` krijgt de waarde 0. De syntax van een Belief Update-regel is als volgt:

```
{ preconditie(s) } Regel { postconditie(s) }
```

Merk op dat de uitvoering van het plan dat de belief update-regel aanroept wacht op het voltooien van de belief update. Als er geen belief aanwezig is dat `oid(N)` kan waarmaken stopt het plan totdat een geschikt belief wordt toegevoegd. Andere plannen en goals die op dat moment actief zijn worden wel verder uitgevoerd.

De postconditie van de belief update-regel bestaat in dit geval uit twee delen, gescheiden door een komma. Het eerste deel verwijdert het oude belief uit de belief base: `not oid(N)`. Het tweede deel voegt een nieuw belief toe: `oid(N+1)`. De preconditie kan ook uit twee delen bestaan. In dat geval worden de operatoren `and`, `or`, `not` gebruikt in plaats van de komma.

De volgende stap in het plan vraagt het zojuist aangemaakte belief op: `B(oid(N))`. De variabele `N` wordt gebruikt in de volgende belief update, waar de nieuwe opdracht in de belief base wordt geplaatst. De preconditie voor deze update is leeg, weergegeven met `true`. De postconditie bevat een nieuw belief `bieding(OID,[])`. Dit betekent dat er in dit belief een lijst met biedingen wordt bijgehouden voor de gegeven `OID`. De lijst is in het begin leeg, aangegeven met `[]` (zoals in Prolog).

Vervolgens wordt een bericht gestuurd aan de vervoerder-agenten met de nieuwe opdracht. Dit gebeurt met de ingebouwde opdracht
`send(Ontvanger,Type,Inhoud)`. De naam die als ontvanger wordt gebruikt moet overeenkomen met de naam die in het definitiebestand
(hier: `definitie_voorbeeld.mas`) aan een van de andere agents gegeven is. Voor het type worden (oa.) `inform` en `request` gebruikt, dat is verder niet

belangrijk. De inhoud is een willekeurig Prolog-predicaat. In dit geval is dat een opdracht in de vorm `opdracht(N,Aantal,Afzender,Bestemming)`.

Als laatste stap in het plan wordt een nieuw doel toegevoegd met de ingebouwde opdracht `adoptz(kiesVervoerder(N))`. Het huidige doel blijft bestaan voor het genereren van nieuwe opdrachten. De agent heeft nu dus twee doelen waarvoor parallel plannen worden uitgevoerd: een doel voor nieuwe opdrachten en een doel om een vervoerder te kiezen voor huidige opdrachten.

Het afhandelen van inkomende berichten gebeurt met PC-rules. Een PC-rule heeft de volgende vorm:

```
plan of bericht <- condities |
{
procedure
...
}
```

Het plan of bericht kan variabelen bevatten die worden gebonden voor gebruik in de procedure. In het geval van een bericht van een vervoerder-agent:

```
message( Vervoerder, inform, La, On, bod(OID,Prijs) ) <- true |
{
NieuwBod(OID,Vervoerder,Prijs)
}
```

De variabelen `La` en `On` (Language en Ontology) worden niet gebruikt in dit voorbeeld, maar wel meegegeven door het platform. Het type `inform` is een constante die wordt gematcht, `Vervoerder` en de twee variabelen in het predicaat `bod(OID,Prijs)` worden gebonden. De conditie is hier leeg. De procedure bestaat uit 1 aanroep naar een belief update-regel: `NieuwBod` met de drie gebonden variabelen. In deze belief update-regel wordt het nieuwe bod toegevoegd aan de lijst met biedingen voor de huidige opdracht. Hiervoor wordt als preconditie van de belief update de huidige lijst opgevraagd (`bieding(OID,L)`) en in de postconditie wordt een nieuw belief toegevoegd met het nieuwe bod vooraan de huidige lijst met de Prolog-operatie `|` in `bieding(OID,[bod(Vervoerder,Prijs)|L])`. Het predicaat bieding bestaat dus uit de variabele `OID` en een lijst van predicaten `bod(X,Y)`.

Het doel `kiesVervoerder(OID)` heeft een PG-rule met preconditie: `biedingGesloten(OID)`. Deze preconditie is een Prologfunctie uit de beliefbase die controleert of er twee biedingen binnen zijn gekomen. De implementatie is `biedingGesloten(OID) :- bieding(OID,[_,_]).` waarbij de _ staat voor een willekeurig atoom of predicaat. Functies en predicaten uit de beliefbase zijn puur Prolog (geen extra 2APL-syntax). Het plan voor `kiesVervoerder` heeft twee stappen: aan aanroep naar de PC-rule `kiesV` en het verwijderen van het doel. In de PC-rule `kiesV` wordt een keuze gemaakt tussen de biedingen uit lijst `L` en worden berichten gestuurd naar de agenten met een toekenning of afwijzing.