Plan Recognition in a High Level Belief-Based Programming Language

Yves Lespérance[a], Alistair Scheuhammer[a], Yu Chen[b], and Petros Faloutsos[a]

[a]York Univ., Toronto (Canada), [b]Univ. of Toronto (Canada)

Motivation

Lots of previous work on plan (and intention) recognition for various types of domains, deterministic and stochastic, discrete and continuous.

But generally restricts the form of plans that observed agent may execute.

Also assumes that plan recognition and the system's response to the observed agent's plan are programmed in independent modules.

Here, we examine how support for plan recognition can be incorporated into an agent programming language, Ergo/Golog, so that the domain, the observed agent's plans, and the system agent's behavior can be specified in the same language.

Ergo/Golog

Ergo [Levesque 19] is a Scheme implementation of Golog agent programming language [Levesque et al 97] based on Situation Calculus.

Programs involve high-level atomic actions that are specified in a declarative action theory.

Ergo supports belief-based programing on top of models with deterministic and stochastic actions and discrete and continuous distributions [Belle & Levesque 15, 18].

Belief update is implemented using sampling and progression (aka particle filtering).

Our Approach to Plan Recognition

Plan recognition as a form of Bayesian inference.

Plan library of observed agent is represented as an Ergo nondeterministic program stored in exoProg fluent.

Have an initial belief distribution.

When an observed agent action is seen, exoProg program is stepped forward to match it and the belief state is updated.

If the action is stochastic, the new belief state is obtained from the prior and the action outcome distribution.

To do this we use a new offlineStepMatch program execution mode that we have implemented.

Programming the System Agent

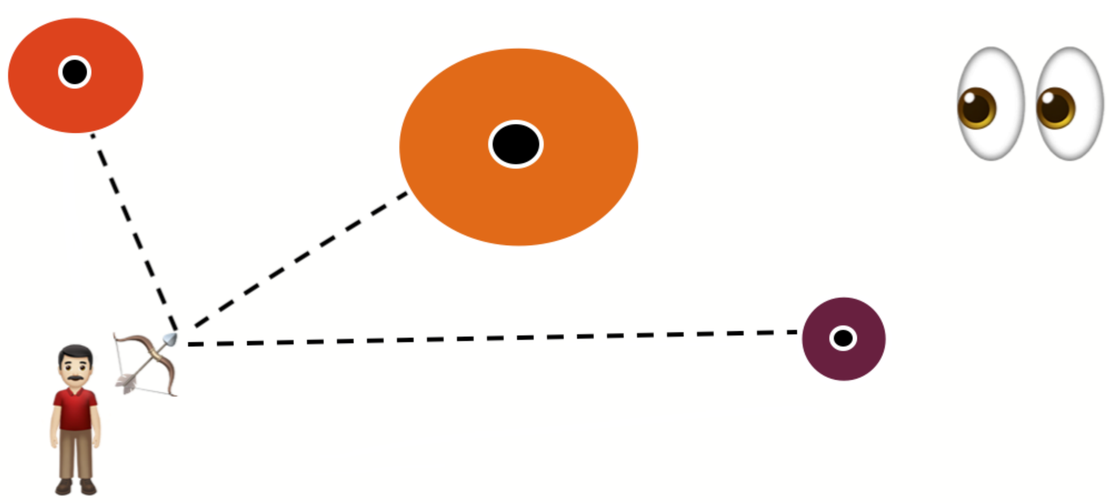
System agent implemented as belief-based program that runs online, receiving observations of observed agent's actions and doing its own.

May be a simple loop that gets an observation of observed agent action, updates beliefs, and then if a trigger condition holds, performs a system agent action, updating beliefs again, and continuing in this way until termination condition comes to hold.

Have implemented a new onlineSynchronized execution mode to support this kind of execution conveniently.

E.g. Target Shooting

Recognize target among 3 possible ones with different angle, size, and value, and 3 different plans, safe, greedy, and optimal, that select distinct targets.



Target Shooting E.g. Implemenation

```
% initial belief state specification: 1M possible states of equal weight
(define-states ((i 1000000))
    decisionPlan 'notDecided          % fluent & value
    decisionTarget 'notDecided
    aim 0.0
    someObservedAim 'notSet
    targetHit (vector #f #f #f)
    halted #f
    exoProg (:begin (:act choosePlan!)                    % plan library
              (:if (eq? decisionPlan 'greedy) (greedy-plan)
                (:if (eq? decisionPlan 'safe) (safe-plan)
                  (optimal-plan)))
              (:choose (:for-some t '(0 1 2) (:act (obsHit! t)))
                (:act obsNoHit!))
              (:act halt!)
            )
)
(define (optimal-plan) (:begin      % other plans are similar but
    (:act chooseTarget!)            % may do setAim! more than once
    (:act setAim!)
    (:starDFS (:act (obsAim! someObservedAim)))
    (:act shoot!)
) ) ...
(define-action choosePlan! #:sequential? #t
    decisionPlan (DISCRETE-GEN 'greedy (/ 1.0 3.0) 'safe (/ 1.0 3.0)
                  'optimal (/ 1.0 3.0))
    exoProg (let ((res (ergo-do #:mode 'offlineStepMatch #:matchAct
                  'choosePlan! exoProg))) (if res (cadr res) :fail))
    weight (if (equal? exoProg :fail) 0.0 weight)
)
(define-action (obsAim! a) #:sequential? #t
    someObservedAim a
    exoProg (let ((res (ergo-do #:mode 'offlineStepMatch #:matchAct
                  (list 'obsAim! a) exoProg))) (if res (cadr res) :fail))
    weight (if (equal? exoProg :fail) 0.0
              (* weight (GAUSSIAN (remainder a 360.0) aim 10.0)))
) ...
```

Target Shooting E.g. Sample Run

```
<<< Exogenous act: choosePlan!
<<< Exogenous act: chooseTarget!
(belief (eq? decisionTarget 0)) returns 0.33029
(belief (eq? decisionTarget 1)) returns 0.329216
(belief (eq? decisionTarget 2)) returns 0.340494
<<< Exogenous act: setAim!
(belief (aiming at target 0)) returns 0.33029
(belief (aiming at target 1)) returns 0.329216
(belief (aiming at target 2)) returns 0.340494
<<< Exogenous act: setAim!     % second setAim! is incompatible with optimal-
(belief (aiming at target 0)) returns 0.5008142458142913     % plan and target 2
(belief (aiming at target 1)) returns 0.4991857541857087
(belief (aiming at target 2)) returns 0.0
<<< Exogenous act: (obsAim! 39.0)
(belief (aiming at target 0)) returns 0.97380791959731
(belief (aiming at target 1)) returns 0.026192080399600744
(belief (aiming at target 2)) returns 0.0
<<< Exogenous act: shoot!
<<< Exogenous act: (obsHit! 1)
(belief (aiming at target 0)) returns 0.0
(belief (aiming at target 1)) returns 1.0
(belief (aiming at target 2)) returns 0.0
```

Other Implemented E.g.'s

Target shooting game where system agent shields target it believes most likely to be aimed at and collects points if it is right.

Marble collection game on 2D grid where observed agent may help or interfere.

Recognizing the goal destination of a robot moving in a 2D continuous space with obstacles with a comparison to [Vered & Kaminka 17].