DIS La Sapienza — PhD Course

Autonomous Agents and Multiagent Systems

Lecture 5: Sensing and Planning under Incomplete Information and Dynamic Evironments

Yves Lespérance

Dept. of Computer Science & Engineering
York University
Toronto, Canada

Embedded Agents

An agent that operates in a real environment (robot or softbot) faces many difficult problems:

- agents planning must be interleaved with its acting, need *incremental execution;*
- agent only has *incomplete knowledge* & must *sense* the environment to know what to do;
- agent operates in a *dynamic environment*; other agents act & agent must detect this & consider how it affects him.

Incremental Execution

Search over nondeterministic program is how Golog/ConGolog support planning.

But search/planning/exploring your options is something you do in your head before you act; at some point, must stop thinking and start acting.

Agent with simple task can do all its planning first, & then execute its plan.

Golog/ConGolog work according to this simple model; search all the way to final situation of nondeterministic program & return the situation; then can execute it.

Incremental Execution (cont.)

But agent that has complex task & must run for a long time cannot do all its planning before it acts.

Must do some planning, then execute some of the plan constructed, then some more planning, then more acting, . . .

For this, simple Golog/ConGolog execution model is inadequate; need a version of the language where *search is interleaved with execution*.

Incomplete Knowledge and Sensing

Another problem: agents have *incomplete knowledge* and must perform *sensing actions*.

E.g. agent must go to the airport & board a flight; cannot know which flight gate to go to in advance; must do sensing once it is at airport to find out!

Representing & reasoning about incomplete knowledge is hard.

Need mechanism to *update knowledge following sensing*.

Golog/ConGolog does not support sensing; Prolog implementation makes closed world assumption.

Planning with Sensing Actions

When plan includes sensing actions, it needs to *branch* on result of sensing.

So in general, need to generate *plans that include sensing actions, branching/conditionals, even loops*.

Very hard search problem!

Can try to avoid generating conditional plans by interleaving sensing and planning; to do this, need search control knowledge.

Operating in a Dynamic World

3rd problem: the world is *dynamic* & other agents perform actions.

Even if agent has complete knowledge initially, does not stay that way.

Need to determine what *exogenous actions* occur & reason about their effects.

Operating in a Dynamic World (cont.)

Sometimes, agent can easily determine what *exogenous actions* have occurred (through sensing); then, executor can monitor for these & incorporate them into the situation.

In general, agent needs to *diagnose* what exogenous actions have occurred to explain sensing data; similar to a planning task (hard).

Execution Monitoring & Replanning

When exogenous action occurs, plan may no longer be valid.

Need to *monitor* plan execution for exogenous actions that make it fail.

When this is detected need to *replan* or do plan repair.

Planning for Dynamic Environment

When planning for dynamic environment, may need to anticipate likely *contingencies*, e.g. action failures, likely events/actions by others, etc.

Contingency plan branches on possible contingencies and achieves goal despite them.

Decision-theoretic planning models uncertain knowledge & action outcome probabilities, & produces plans that maximize expected utility.

Multiagent Planning

Other intelligent agents are *rational*: will not do actions that do not further their goals; also reason about what other agents may do.

Game-theoretic planning finds optimal strategies for agents that interact with other agents.

IndiGolog

IndiGolog [DegLev99a] addresses some of these problems; supports:

- *interleaving search & execution*: use *search block* when lookahead/planning is needed; otherwise makes arbitrary choice of next action & executes;
- *execution of sensing actions* & knowledge expansion by sensed information; uses a *dynamic closed world assumption*;
- *observation of exogenous actions* (user must define monitoring routines).

IndiGolog Search Block

By default, IndiGolog does no search/lookahead.

But, programmer can tell interpreter to *search* over block of code (on-line) using new *search block* construct $\Sigma\ \delta$.

Semantics :

$$Trans(\Sigma\ \delta, s, \delta', s') \equiv$$
$$Trans(\delta, s, \delta', s') \wedge \exists \delta'', s''\ Do(\delta', s', \delta'', s'')$$

Can cache the plan found for efficiency.

IndiGolog Sensing

Program can include *sensing actions* that acquire new information.

Sensed fluent axioms specify what condition is sensed, e.g.

$$SF(senseDoor(d), s) \equiv Open(d, s)$$

Programmer must provide method to get sensing result.

Result of sensing is *added* to basic action theory (& assumed consistent).

IndiGolog Semantics

A history σ is a sequence of ground actions with associated sensing results.

An online configuration (δ_i, σ_i) involves a program & a history.

Can perform an *online transition* $(\delta_i, \sigma_i) \rightarrow (\delta_{i+1}, \sigma_{i+1})$ iff

$$\mathcal{D} \cup \{Sensed[\sigma_i]\} \models Trans(\delta_i, end[\sigma_i], \delta_{i+1}, end[\sigma_{i+1}]),$$

where σ_{i+1} is σ_i if transition does not do an action, & $\sigma_i \circ (a, x)$ if it performs action a with sensing result x.

An online configuration (δ_n, σ_n) can *successfully terminate* iff

$$\mathcal{D} \cup \{Sensed[\sigma_n]\} \models Final(\delta_n, end[\sigma_n]).$$

IndiGolog Implementation

In Prolog implementation, evaluation of projection queries uses regression, but traps on matching sensing results.

This amounts to making *dynamic closed world assumption* [DegLev99b].

If program never tests an initially unknown condition before sensing it, then it will never get an incorrect answer (from CWA).

IndiGolog Exogenous Actions

Changes in environment can be detected as *exogenous actions:*

Interpreter monitors for them and adds them to history.

Programmer must provide method to check for their occurrence.

Effects must be specified as for ordinary actions.

IndiGolog Replanning

When an exogenous action happens while executing a search block, may need to *replan* [DRS98].

E.g. when running mail delivery program that minimizes distance travelled and new shipment order is made.

Then, IndiGolog checks if the sequence of actions found earlier is still an execution of the program in the search block; otherwise, it redoes the search.

Original IndiGolog cannot find a plan for this e.g. because it restarts search from remaining program.

IndiGolog of [LesNg00] restarts search from original program, so it does find a new plan.

Only committed to the actions it has performed.

Reasoning about Incomplete Knowledge and Action

Dynamic CWA is not always warranted.

Reasoning with arbitrary incomplete KBs is intractable.

Some work on KBs with limited forms of incompleteness where reasoning is efficient, e.g. [BacPet98], [LiuLev05].

Possible Values Implementation of IndiGolog

Proposed in [SarVas05].

Incomplete knowledge restricted to having a set of *possible values for each functional fluent*, e.g.

$$temp(S_0) = 19 \ \vee \ temp(S_0) = 20 \ \vee \ temp(S_0) = 22$$

A formula is *possibly true* if there exists a choice of possible values for the fluents in it that makes it true.

A formula ϕ is *certainly true/known to be true* iff $\neg phi$ is not possibly true.

Possible Values Implementation of IndiGolog (cont.)

Handles sensing actions such that if we get result r, then the value of the fluent f must be/cannot be v, when w is known to be true.

Regression mechanism is defined; guaranteed to be sound under some conditions.

Contingent Planning for APLs [LDO07]

Assume that both planning agent's task δ and behavior of agents in environment ρ are expressed as high-level nondeterministic concurrent programs in ConGolog (or some other APL); environment has higher priority.

Planning must produce deterministic conditional plan that can be successfully executed *against all possible executions of environment program*.

Handle actions with nondeterministic effects & sensing actions by treating them as actions that trigger an environmental reaction that is not under agent's control.

E.g. An Interfering Environment Agent

IA moves stacked blocks back to table:

proc $interferingAgtBehavior(IA, n)$
  $(n \leq 0?)|$
  $((n > 0 \wedge LastActionNotBy(IA) \wedge \exists x, yOn(x, y))?$;
    $[\pi\, x.\exists yOn(x, y)?; moveToTable(IA, x);$
      $interferingAgtBehavior(IA, n - 1)]$ |
    $[noOp(IA);\ interferingAgtBehavior(IA, n)])$
endProc

n is bound on number of interfering moves.

E.g. A Planning Agent

PA's task is to build 3 blocks tower:

proc $mkTower(PA)$
  while $\neg HaveTower$ **do**
    if $\exists x, y\, On(x, y)$ **then**
      $\pi\, x, z.[\exists y\, On(x, y)?; move(PA, z, x)]$
    else
      $\pi\, x, y.move(PA, x, y)$
    endIf
  endWhile
endProc

Can vary amount of nondeterminism in task spec.

E.g. Actions with Nondeterministic Effects

Nature agent NA determines whether move attempts succeed:

proc $natureBehavior(NA, n)$
  $\pi\, x, y.[(n > 0 \wedge MoveAttempted(PA, x, y))?;$
    $moveFails(NA, PA, x, y);$
    $natureBehavior(NA, n - 1)] \mid$
  $\pi\, x, y.[MoveAttempted(PA, x, y)?;$
    $moveSucceeds(NA, PA, x, y);$
    $natureBehavior(NA, n)]$
endProc

n is bound on number of failures.

E.g. Sensing

PA does sensing by querying humidity sensor agent HSA:

proc $humiditySensorBehavior(HSA)$
  $\langle\, x : WetnessQueried(PA, HSA, x) \rightarrow$
    $(reportWet(HSA, PA, x) \mid$
    $reportNotWet(HSA, PA, x))\, \rangle$
endProc

Sensor report has knowledge producing effect.

In general, learn that preconditions of observed exogenous actions must hold.

E.g. Helpful Environment Agent

Environment agent DA will dry a block when requested:

proc $dryingAgtBehavior(DA)$
$\langle\ x : DryingRequested(PA, DA, x) \rightarrow dry(DA, x)\ \rangle$
endProc

Contingent Planning – APL Primitives

- *EnvTrans*$(\langle \rho, s \rangle, \langle \rho', s' \rangle)$: agent *considers it possible* that environment program ρ in state s can make transition to state s' with program ρ' remaining;

- *AgtTrans*$(\langle \delta, s \rangle, \langle \delta', s' \rangle)$: agent *knows* that agent program δ in state s can make transition to state s' with program δ' remaining;

- *AgtFinal*$(\langle \delta, s \rangle)$: agent *knows* that agent program δ can legally terminate in state s.

Contingent Planning – Definition

AbleBy$(\sigma, \delta, \rho, s)$ means that agent is able to execute task δ in environment ρ in state s by executing conditional program σ.

AbleBy$(\sigma, \delta, \rho, s)$ is smallest relation $\mathcal{R}(\sigma, \delta, \rho, s)$ s.t.:

(A) for all triples (δ, ρ, s), if
EnvBlocked$(\langle \rho, s \rangle)$ and *AgtFinal*$(\langle \delta, s \rangle)$,
then $\mathcal{R}(nil, \delta, \rho, s)$;

Contingent Planning – Definition (cont.)

(B) for all quadruples $(\sigma, \delta, \rho, s)$, if
EnvBlocked$(\langle \rho, s \rangle)$ and
there exist δ', s' such that
AgtTrans$(\langle \delta, s \rangle, \langle \delta', s \rangle)$ and $\mathcal{R}(\sigma, \delta', \rho, s')$ and
actsPerf$(s, s') = \langle\rangle$,
then $\mathcal{R}(\sigma, \delta, \rho, s)$;

(C) for all $a, \sigma, \delta, \rho, s$, if
EnvBlocked$(\langle \rho, s \rangle)$ and
there exist δ', s' such that
AgtTrans$(\langle \delta, s \rangle, \langle \delta', s' \rangle)$ and $\mathcal{R}(\sigma, \delta', \rho, s')$ and
actsPerf$(s, s') = a$,
then $\mathcal{R}((a; \sigma), \delta, \rho, s)$;

Contingent Planning – Definition (cont.)

(D) for all triples (δ, ρ, s), if
it is not the case that *EnvBlocked*$(\langle \rho, s \rangle)$ and
FiniteEnvTrans$^*(\langle \rho, s \rangle)$ and
for all ρ', s'
$EnvTrans^m(\langle \rho, s \rangle, \langle \rho', s' \rangle)$ implies
for some σ', $\mathcal{R}(\sigma', \delta, \rho', s')$,
then $\mathcal{R}(\sigma, \delta, \rho, s)$ where
$\sigma =$ **if** $Done($*actsPerf*$(s, s_1))$ **then** σ_1 **else**
$\ldots$
if $Done($*actsPerf*$(s, s_n))$ **then** σ_n **else** nil
and $EnvTrans^m(\langle \rho, s \rangle) = \{\langle \rho_1, s_1 \rangle, \ldots, \langle \rho_n, s_n \rangle\}$
and $\mathcal{R}(\sigma_i, \delta, \rho_i, s_i)$ for $i = 1, \ldots, n$.

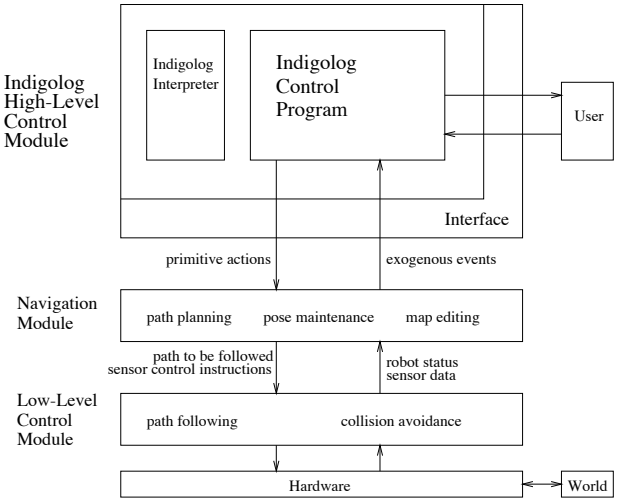
Contingent Planning Implementation

Relies on possible values implementation of IndiGolog [SarVas05].

Handles knowledge producing effects of observing environment actions of the form: exogenous action a is possible iff fluent f has/does not have value v, when condition w holds.

Implemented by adapting [SarVas05]'s regression mechanism.

Interfacing with a Robot Control Architecture E.g. [LTJ99]



Interfacing High-Level Controller with Rest of Architecture

One approach: view rest of architecture as another agent; primitive actions are commands to it and signals it sends back are exogenous actions.

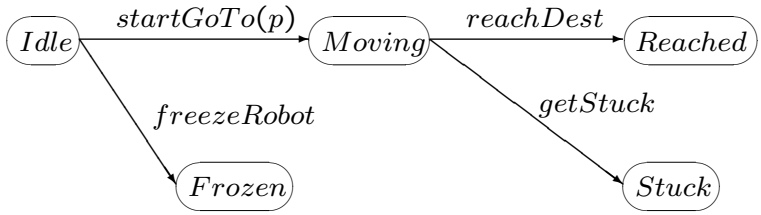
In high-level controller, use model of rest architecture.

Simple version of this in [LTJ99].

Other work on this problem: [FinPir01], and [GroLak00] on cc-Golog for continuous control.

Interfacing H-L Controller, E.g. model of [LTJ99]

From navigation point of view, model robot as being in a state $robotState(s)$ which can change as a result of actions by the high-level controller and exogenous events:



Also action $resetRobot$ that returns robot to $Idle$ state; can be performed in any state. Also fluents $robotDestination(s)$ and $robotPlace(s)$.

Interfacing H-L Controller [LTJ99] (cont.)

So treat "going to a location" as an "activity" in the sense [Gat92]; the primitive actions are not the activity; only initiate and terminate it.

SitCalc-Based Robot Control Work

Controllers written in situation calculus-based high-level programming languages tested on real robots at York, U. of Toronto, and U. of Bonn; Bonn group created very successful “museum guide” application [Bur+98].

At York: high-Level controllers that do mail delivery and handle new orders and navigation failures; run on RWI B12 and Nomad Super Scout.

Use low-level control and navigation modules based on software developed for ARK project [Nic+98]. Modules run in separate processes that communicate via TCP/IP sockets.

In [LesNg00], system using extended IndiGolog tested on scenarios that require planning to optimize delivery route, reacting/replanning when new order arrives or navigation fails.

Recap

High-level programming in situation calculus is promising approach to agent programming.

IndiGolog high-level programming language supports:

- complex behaviors: loops, concurrency, etc.,
- reactive behaviors: interrupts,
- on-line linear planning,
- execution in dynamic & incompletely known environments with dynamic CWA.

Have extensions for efficient reasoning with limited forms of incomplete knowledge and sensing, contingent planning, etc.

Recap (cont.)

Have interfaces to OAA [LapLes02] & JADE with some support for FIPA ACL & protocols [MarLes04a][MarLes04b].

Logical foundations support:

- formal specification and verification,
- general reasoning with *incomplete* information.

Tested in applications including real robot control.

Also high-level programming in event calculus [ShaWit00] and fluent calculus [Thielscher00], and related work on Model-Based Programming [WCG01] and structured-reactive controllers [Beetz01].

Major Issues in Current Agent Research

Efficient reasoning about knowledge and action [BacPet98], [DemDel00], [LiuLev05], progression [Thielscher05], [VasLev07].

Models of planning with sensing [BacPet98], [FPR00], [Sardina01], [Reiter01a], [Levesque96], [LLLS01], [DLLS04], [SarPad07], [SDLL07] and contingent planning [LesNg00], [LDO07] for agents.

Interfacing with state-of-the-art operator-based planners [BFM07], [CHL07].

Programmer-friendly implementation, tools.

APLs with declarative goals [VVM03], [SarPad07], models of goals and goal change [SLL07].

Major Issues in Current Agent Research (cont.)

Agent programming languages based on probability theory [FinPir01], [GroLak01], [BHL95] and decision theory [BRST00], [WCG01], and reinforcement learning mechanisms for these.

Reasoning about other agents, their knowledge, goals, etc. [SLL02], [KhaLes05], [SLL07]; finding efficient methods for this.

Game-theoretic planning [FinLuk04], [FFL07], [GLL07].

Plan, activity, and intent recognition [GouLes07].

Other Issues in Agent Research

Dealing with the ramification and qualification problems [Reiter01b].

Modeling & interfacing with non-symbolic processes, dealing with temporal constraints [Reiter98], continuous control [GroLak00], [Sandewall97].

Accounts of sensing and knowledge change [Iocchi99], [ShaWit00], [MciSch00], on-line sensors and just-in-time programs [DLS01], noisy sensors [BHL95], models of vision, anchoring [CorSaf01].

Other Issues in Agent Research (cont.)

Execution monitoring and replanning [DRS98], agent architecture.

Practical use of planning as nondeterministic programming.

MAS development tools and methodologies.

Verification.

Emotions, Personality, etc.