On the Semantics of Deliberation in IndiGolog — From Theory to Implementation

Yves Lespérance

Department of Computer Science & Engineering
York University
Toronto, Canada

Joint work with Giuseppe De Giacomo, Hector Levesque, and Sebastian Sardina.

See paper in *Annals of Math. and AI*, **41**(2-4), 259–299, Aug. 2004.

`http://www.cse.yorku.ca/~lesperan`

The Problem: What is Planning under Incomplete Information?

Interested in planning where agent only has *incomplete information* and run-time *sensing* is required.

Needed in many applications, e.g., information agents, autonomous robots, etc.

Need specification of what it means to solve the problem.

What should planner return?

Can’t ask for straight-line plan, because often none is a solution.

Could say must return a program, but in general, figuring out how to execute a program can be as hard as planning (e.g., Golog programs).

The Problem (cont.)

Planning should produce specification of the desired behavior that does not require deliberation to interpret.

In [Levesque 96], problem addressed by requiring that the planner return programs of a syntactically restricted form. Also the case in most work on planning with incomplete information.

Our Account of Planning under Incomplete Information

Here: a more abstract, semantically-motivated account.

Treat plans as *epistemically feasible* programs, i.e., programs for which the agent, at every stage of execution, by virtue of what it knew initially and the subsequent sensing, always *knows* what step to take next.

Our Account (cont.)

Account applies not only to planning but also to execution of *high-level programs* in agent programming language such as Golog.

High-level programs can be non-deterministic; then interpreter must search for an execution.

Planning is a special case.

Can specify domain-specific control information, plan skeletons, etc.

IndiGolog = Golog + concurrency + incremental execution with sensing. Supports lookahead search/planning over parts of program.

Our Account (cont.)

Account framed as a semantics for search construct of IndiGolog.

Resolves problems with semantics in [De Giacomo & Levesque 99]:

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

Agent may know that there is some action that leads to successful termination without knowing which.

E.g. $\Sigma([a;$ **if** ϕ **do** b **else** c **endIf**$] \mid d)$ may do a even when agent does not know whether ϕ after doing a.

Example: Getting on Flight at Airport [Levesque 96]

Agent wants to board flight. Does not know gate in advance; must sense at airport.

To do planning to get on flight, execute IndiGolog program:

$\Sigma(achieve(On\_plane(Flight123), True))$

where

$achieve(Goal, GoodSit) \stackrel{\text{def}}{=}$

while $\neg Goal$ **do**

$\pi\, a[a; GoodSit(now)?]$

endWhile

Example: Getting on Flight (cont.)

A fully detailed plan/solution:

$go(Airport)$;
$check\_departures$;
if $Parked(Flight123, GateA)$ **then**
    $go(GateA)$; $board\_plane(Fligth123)$
else
    $go(GateB)$; $board\_plane(Fligth123)$
endIf

Without sensing, goal cannot be achieved!

Our Account (cont.)

- characterizes epistemically feasible programs formally,
- defines IndiGolog search construct in terms of these,
- shows that programs belonging to certain syntactically restricted classes are epistemically feasible,
- shows that in certain cases — conformant planning and conditional planning — when there is an epistemically feasible program that solves the problem, there must be a program from a syntactically restricted class that does too, and
- specifies a form of deliberation with execution monitoring and replanning.

Situation Calculus [McCarthy63]

A language of predicate logic for representing dynamically changing worlds.

The constant S_0 represents the initial situation.

The term $do(\alpha, s)$ represents the situation that results from primitive action α being performed in situation s.

Predicates and functions whose value varies from situation to situation are called fluents.

e.g.

$$At(Home, S_0)$$

$$At(Airport, do(go(Airport), S_0))$$

Modeling Knowledge and Sensing

Model knowledge by adapting Kripke semantics to situation calculus [Moore 85]:

$$\textbf{Know}(agt, \phi(now), s) \stackrel{\text{def}}{=} \forall s'(K(agt, s', s) \supset \phi(now/s')).$$

Here, binary sensing actions only.

Information provided by sensing action represented by $SF(a, s)$. Have axioms:

$$SF(sense_\phi, s) \equiv \phi(s)$$

$$SF(nonSensingAct, s) \equiv True$$

Modeling Knowledge and Sensing (cont.)

Knowledge dynamics specified by successor state axiom for K [Scherl & Levesque 93, Levesque 96]:

$$
\begin{aligned}
&K(s'', do(a, s)) \equiv \\
&\quad \exists s'[K(s', s) \wedge s'' = do(a, s') \wedge Poss(a, s') \wedge \\
&\qquad (SF(a, s') \equiv SF(a, s))]
\end{aligned}
$$

After sensing, agent knows truth-value of associated fluent.

Specifying Domain Dynamics in Situation Calculus

Use theory of the form:

- axioms describing initial situation S_0,
- action precondition axioms that characterize $Poss(a, s)$,
- successor state axioms that characterize $F(\vec{x}, do(a, s))$ in terms of what holds in s — provide solution to frame problem [Reiter 91],
- sensed fluent axioms that characterize $SF(a, s)$,
- successor state axiom for K,
- unique names axioms for primitive actions,
- foundational axioms [Lakemeyer & Levesque 98, Reiter 01].

IndiGolog Constructs

α,	primitive action
ϕ?,	test/wait for a condition
$(\delta_1; \delta_2)$,	sequence
if ϕ **then** δ_1 **else** δ_2 **endIf**,	conditional
while ϕ **do** δ **endWhile**,	loop
proc $\beta(\vec{x})$ δ **endProc**,	procedure definition
$\beta(\vec{t})$,	procedure call
$(\delta_1 \mid \delta_2)$,	nondeterministic choice of action
$\pi\, \vec{x}\, [\delta]$,	nondet. choice of arguments
δ^*,	nondeterministic iteration
$(\delta_1 \parallel \delta_2)$,	concurrent execution
$(\delta_1 \rangle\!\rangle \delta_2)$,	concurrency with priorities
$\delta^{\parallel}$,	concurrent iteration
$\langle \vec{x} : \phi \rightarrow \delta \rangle$,	interrupt
$\Sigma(\delta)$,	search block

Semantics

Based on transition systems.

$Trans(\delta, s, \delta', s')$ means that can make a transition

$$(\delta, s) \rightarrow (\delta', s')$$

by executing a single primitive action or test.

$Final(\delta, s)$ means that in configuration (δ, s), computation can terminate.

Defined by axioms, e.g.:

$$Trans(\alpha, s, \delta, s') \equiv$$
$$Poss(\alpha, s) \wedge \delta = nil \wedge s' = do(\alpha, s)$$

$$Trans([\delta_1; \delta_2], s, \delta, s') \equiv$$
$$Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta, s') \quad \vee$$
$$\exists \delta'.\delta = (\delta'; \delta_2) \wedge Trans(\delta_1, s, \delta', s')$$

Semantics (cont.)

$Do(\delta, s, s')$ means there is an execution of program δ that starts in situation s and terminates in s'. Formally:

$$Do(p, s, s') \stackrel{\text{def}}{=} \exists p'.Trans^*(p, s, p', s') \wedge Final(p', s'),$$

where $Trans^*$ is the reflexive transitive closure of $Trans$.

An *offline execution* of program p from situation s is sequence of actions $a_1, \ldots, a_n$ such that:

$$Axioms \models Do(p, s, do(a_1, \ldots, do(a_n, s)))$$

Offline executor similar to compiler; no access to sensing results!

Epistemically Accurate Theories

Theories where what is known accurately reflects what the theory says about the system, essentially:

- Initial situation characterized by an axiom of the form **Know**(ϕ_0, S_0) where ϕ_0 is an *objective formula*.
- Accessibility relation K is reflexive in all situations.
- Finite set of action types.
- No functional fluents, standard names for objects, and theory decides all equality sentences that don't involve program terms.

Epistemically Accurate Theories (cont.)

For epistemically accurate theories we have:

Theorem 1 *For any objective sentence about situation* s, $\phi(s)$, $Axioms \cup \{Sensed[\sigma]\} \models \phi(end[\sigma])$ *if and only if* $Axioms \cup \{Sensed[\sigma]\} \models$ **Know**$(\phi, end[\sigma])$.

So if some objective property of the system is entailed, then it is also known and vice-versa.

Epistemically Feasible Deterministic Programs

Programs that are deterministic and for which executor always has enough information to continue the execution, always knows what next step is. Formally:

$$EFDP(dp, s) \stackrel{\text{def}}{=} \forall dp', s'.Trans^*(dp, s, dp', s') \supset LEFDP(dp', s').$$

i.e., a program is an $EFDP$ in a situation if all reachable configurations involve a locally epistemically feasible deterministic program ($LEFDP$).

Epistemically Feasible Deterministic Programs (cont.)

Locally epistemically feasible deterministic programs:

$$LEFDP(dp, s) \stackrel{\text{def}}{=}$$
$$\textbf{Know}(Final(dp, now) \wedge \neg\exists dp', s'.Trans(dp, now, dp', s'), s) \vee$$
$$\exists dp'.\textbf{Know}(\neg Final(dp, now) \wedge UTrans(dp, now, dp', now), s) \vee$$
$$\exists dp', a.\textbf{Know}(\neg Final(dp, now) \wedge UTrans(dp, now, dp', do(a, now)), s)$$

$$UTrans(dp, s, dp', s') \stackrel{\text{def}}{=} Trans(dp, s, dp', s') \wedge$$
$$\forall dp'', s''.Trans(dp, s, dp'', s'') \supset dp'' = dp' \wedge s'' = s'$$

i.e., a program is a $LEFDP$ in a situation if the agent knows that it is currently $Final$ or knows what unique transition it can perform next.

Epistemically Feasible Det. Programs — Examples

Our original detailed plan to get on a flight is an $EFDP$:

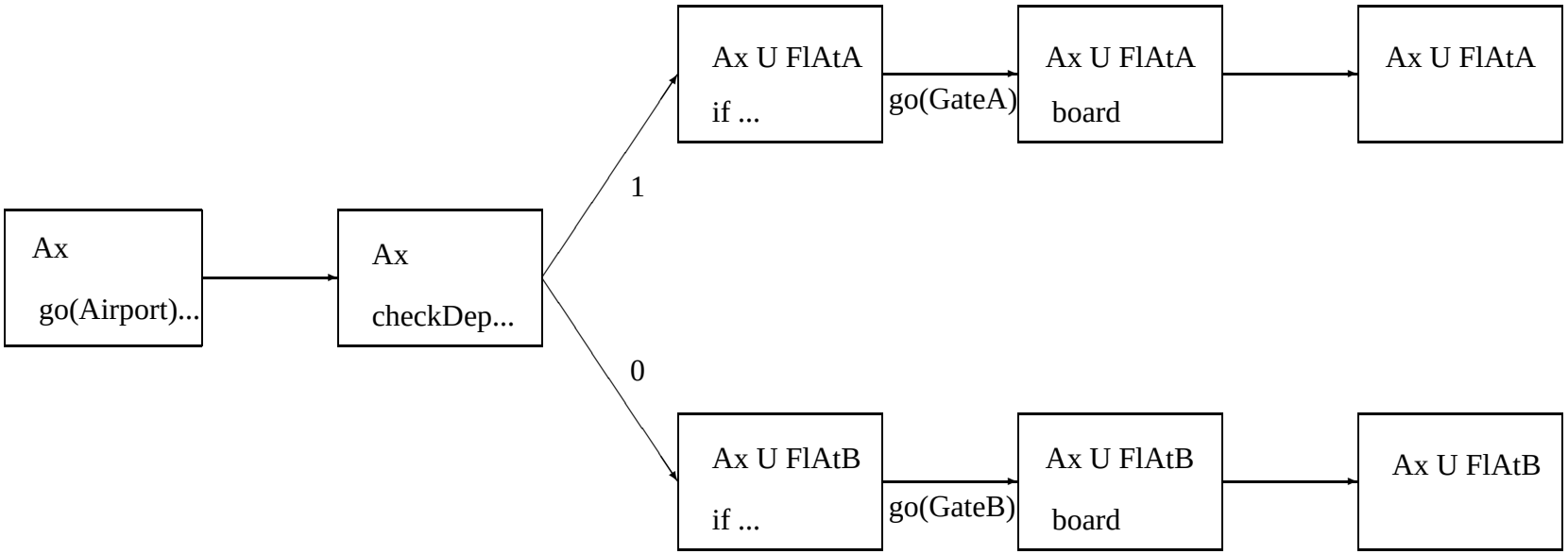
$go(Airport)$;
$check\_departures$;
if $Parked(Flight123, GateA)$ **then**
    $go(GateA)$; $board\_plane(Fligth123)$
else
    $go(GateB)$; $board\_plane(Fligth123)$
endIf

But if delete the $check\_departures$ sensing action, no longer an $EFDP$.
Agent does not know what to do next at the test.

Online Execution

Plans that do sensing are meant to be executed *online*, with sensing results being added to the domain theory.

E.g. for “get on flight 123” program:



Formalizing Online Executions

A *history* describes a run with actions and their sensing results, e.g.:

$$(go(Airport), 1) \cdot (check\_departures, 0) \cdot (go(GateB), 1).$$

$end[\sigma]$ stands for the *end situation* of history σ, e.g.:

$$do(go(GateB), do(check\_departures, do(go(Airport), S_0))).$$

$Sensed[\sigma]$ stands for formula specifying the *sensing results* of history, e.g.:

$$SF(go(Airport), S_0) \wedge$$
$$\neg SF(check\_departures, do(go(Airport), S_0)) \wedge$$
$$SF(go(GateB), do(check\_departures, do(go(Airport), S_0)))$$

Formalizing Online Executions (cont.)

An *online execution* of a program p starting from a history σ wrt a model M of $Axioms \cup \{Sensed[\sigma]$ is a sequence $(p_0 = p, \sigma_0 = \sigma), \ldots$ such that for $i = 0, \ldots$:

$$Axioms \cup \{Sensed[\sigma_i]\} \models Trans(p_i, end[\sigma_i], p_{i+1}, end[\sigma_{i+1}])$$

$$\sigma_{i+1} = \begin{cases} \sigma_i & \text{if } end[\sigma_{i+1}] = end[\sigma_i] \\ \sigma_i \cdot (a, \mu) & \text{if } end[\sigma_{i+1}] = do(a, end[\sigma_i]) \text{ and} \\ & \mu = 1 \text{ if } M \models SF(a, end[\sigma_n]) \text{ else } \mu = 0 \end{cases}$$

Successfully terminates if sequence is finite, ending with (p_n, σ_n), and

$$Axioms \cup \{Sensed[\sigma_n]\} \models Final(p_n, end[\sigma_n]).$$

Epistemically Feasible Det. Programs — Properties

An $EFDP$ need not always terminate.

But if it is entailed that an $EFDP$ will terminate, then it can be successfully executed online whatever the sensing outcomes may be:

Theorem 2
Let dp be such that $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dp, end[\sigma])$. Then, $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dp, end[\sigma], s_f)$ if and only if for each model M of $Axioms \cup \{Sensed[\sigma]\}$, the complete online execution of dp from σ wrt M successfully terminates.

New Search Operator — Semantics

$$Trans(\Delta_e(p), s, dp', s') \equiv$$
$$\exists dp.EFDP(dp, s) \wedge$$
$$\exists s_f.Trans(dp, s, dp', s') \wedge Do(dp', s', s_f) \wedge Do(p, s, s_f).$$
$$Final(\Delta_e(p), s) \equiv Final(p, s).$$

Thus $Axioms \cup \{Sensed[\sigma]\} \models Trans(\Delta_e(p), s, dp', s')$ iff axioms entail that there is some $EFDP$ dp that reaches a $Final$ situation of the original program p no matter how sensing turns out (i.e., in every model).

Note: commits to the selected $EFDP$.

New Search Operator — Properties

Theorem 3 *If* $Axioms \cup \{Sensed[\sigma]\} \models Trans(\Delta_e(p), end[\sigma], p', s')$, *then*

1. $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(p, end[\sigma], s_f)$,
   i.e., program in search block reaches a $Final$ *situation in every model,*

2. $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(\Delta_e(p), end[\sigma], s_f)$,
   i.e., so does $\Delta_e(p)$, *and*

3. *For each model* M *of* $Axioms \cup \{Sensed[\sigma]\}$, *all online executions from* $(\Delta_e(p), \sigma)$ *wrt* M *successfully terminate,*
   i.e., $\Delta_e(p)$ *can be successfully executed online whatever the sensing results are.*

Syntax-Based Accounts of Deliberation

In general, search/deliberation is very hard; amounts to planning where class of potential plans is very general.

Two interesting restricted classes:

- programs that do not perform sensing, i.e., *conformant plans*, and

- programs that are guaranteed to terminate in a bounded number of steps, i.e., *conditional plans*.

Have shown that for these 2 classes, one can restrict attention to simple syntactically-defined classes of programs without loss of generality.

Tree Programs

Class of *(sense-branch) tree programs* $TREE$ defined by:

$$dpt ::= nil \mid False? \mid a; dpt_1 \mid True?; dpt_1 \mid sense_\phi; \textbf{ if } \phi \textbf{ then } dpt_1 \textbf{ else } dpt_2$$

where a is non-sensing action, and $dpt_1, dpt_2 \in TREE$.

Includes conditional programs where tests only involve *conditions that have just been sensed* (or trivial tests).

Tree Programs — Properties

Whenever such a tree program is executable, it is also epistemically feasible — agent will always know what to do next:

Theorem 4 *Let dpt be a tree program, i.e., $dpt \in TREE$. Then, for all histories σ, if $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpt, end[\sigma], s_f)$ then $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dpt, end[\sigma])$.*

Also, by Theorem 2, under the conditions above, all online executions of (dpt, σ) are terminating.

Finding a tree program that yields an execution of a program in a search block is our analogue of conditional planning (under incomplete information).

Tree Programs Can Express Any Bounded Strategy

Can show that tree programs are sufficient to express any strategy where there is a known bound on the number of steps it needs to terminate.

For any $EFDP$ for which this condition holds, there is a tree program that produces the same executions:

Tree Programs Can Express Bounded Strategy (cont.)

Theorem 5 *For any program dp that is*

1. *an epistemically feasible deterministic program, i.e., $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dp, end[\sigma])$ and*

2. *such that there is a known bound on the number of steps it needs to terminate, i.e., where there is an n such that $Axioms \cup \{Sensed[\sigma]\} \models \exists p', s', k.k \leq n \wedge Trans^k(dp, end[\sigma], p', s') \wedge Final(p', s')$,*

there exists a tree program $dpt \in TREE$ such that for each model M of $Axioms \cup \{Sensed[\sigma]\}$ the complete execution of dpt from σ wrt M successfully terminates in the same history as dp from σ wrt M.

Tree Programs Can Express Bounded Strategy (cont.)

So if restrict attention to $EFDP$s that terminate in bounded number of steps, then can further restrict attention to programs of very specific syntactic form, without any loss in generality.

Could be used to simplify implementation of deliberation.

Linear Programs

Define class of *linear programs* $LINE$ by:

$$dpl ::= nil \mid a; dpl_1 \mid True?; dpl_1$$

where a is non-sensing action, and $dpl_1 \in LINE$.

Only includes sequences of actions or trivial tests.

So whenever such a plan is executable, then it is also epistemically feasible — agent always knows what to do next:

Theorem 6 *Let dpl be a linear program, i.e., $dpl \in LINE$. Then, for all histories σ, if $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpl, end[\sigma], s_f)$ then $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dpl, end[\sigma])$.*

By Theorem 2, under these conditions, all online executions of (dpl, σ) are terminating.

Linear Programs Can Express Any Strategy With No Sensing

For any $EFDP$ not performing sensing, there is a linear program that produces the same execution:

Theorem 7 *For any dp that does not include sensing actions, such that $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dp, end[\sigma])$, there exists a linear program dpl such that for each model M of $Axioms \cup \{Sensed[\sigma]\}$ the complete execution of dpt from σ wrt M successfully terminates in the same history as dp from σ wrt M.*

If domain has no sensing actions, then linear programs are sufficient to express every strategy.

E.g. Implementation: Search Operator Looking for Tree Program

```
trans(search_t(P),H,DPT1,H1):-
      buildTree(P,DPT,H), trans(DPT,H,DPT1,H1).
buildTree(P,[],H):- final(P,H).
buildTree(P,[(true)?|DPT],H):-
     trans(P,H,P1,H), buildTree(P1,DPT,H).
buildTree(P,[A,if(F,DPT1,DPT2)]):-
     trans(P,H,P1,[(A,_)|H]), senses(A,F),
     buildTree(P1,DPT1,[(A,1)|H]),
     buildTree(P1,DPT2,[(A,0)|H]).
buildTree(P,[A|DPT],H):-
     trans(P,H,P1,[(A,_)|H]), not senses(A,_),
     buildTree(P1,DPT,[(A,1)|H]).
buildTree(P,(false)?,H):- inconsistent(H).
```

Sound, but not complete.

Dealing with Non-Binary Sensing Actions

Our account handles sensing actions with non-binary, but finitely many outcomes. What about the general case?

Even for finite number of sensing outcomes, conditional planning is impractical without advice from programmer as to what conditions the plan should branch on.

[Sardina 01] develops an implementation of search in IndiGolog that uses such information.

Execution Monitoring

$\triangle_e$ commits to a particular $EFDP$. Bad idea if exogenous actions can make the selected $EFDP$ impossible to execute.

Should define a search operator that monitors the execution of the selected $EFDP$ and replans when necessary as in [De Giacomo *et al* 98][Lespérance & Ng 00].

Execution Monitoring (cont.)

Can define a search operator that monitors the execution of the selected $EFDP$ and replans when necessary:

$$Trans(\Delta_{em}(p), s, p', s') \equiv$$
$$\exists dp, dp'.p' = \textbf{mnt}(dp', s', p, s) \wedge EFDP(dp, s) \wedge$$
$$\exists s_f.Trans(dp, s, dp', s') \wedge Do(dp', s', s_f) \wedge Do(p, s, s_f)$$

$$Final(\Delta_{em}(p), s) \equiv Final(p, s)$$

mnt is new monitoring construct.

Execution Monitoring (cont.)

$$\textbf{ptb}(\textbf{mnt}(dp, s_e, p_i, s_i), s) \equiv$$
$$s_e \neq s \wedge \neg\exists s_f[Do(dp, s, s_f) \wedge Do(p_i \| P_{exo}), s_i, s_f)]$$

dp has been perturbed in s if expected sit s_e is different and we can no longer execute dp to a final situation of original program p_i; P_{exo} is $(\pi a.Exo(a)?; a)^*$.

$$\textbf{rcv}(\textbf{mnt}(dp, s_e p_i, s_i), s, dp_r) \equiv$$
$$\exists p_i', s_f.Trans^*((p_i \| P_{exo}), s_i, (p_i' \| P_{exo}), s) \wedge$$
$$EFDP(dp_r, s) \wedge \exists s_f.Do(dp_r, s, s_f) \wedge Do(p_i', s, s_f)$$

A recovered plan dp_r is an $EFDP$ that can solve the original program p_i while accounting for every action executed so far.

Execution Monitoring (cont.)

$$
\begin{aligned}
&Trans(\textbf{mnt}(dp, s_e, p_i, s_i), s, p', s') \equiv \\
&\quad \neg\textbf{ptb}(\textbf{mnt}(dp, s_e, p_i, s_i), s) \wedge \\
&\qquad \exists dp'.p' = \textbf{mnt}(dp', s', p_i, s_i) \wedge Trans(dp, s, dp', s') \vee \\
&\quad \textbf{ptb}(\textbf{mnt}(dp, s_e, p_i, s_i), s) \wedge \exists dp_r.\textbf{rcv}(\textbf{mnt}(dp, s_e, p_i, s_i), s, dp_r) \wedge \\
&\qquad \exists dp'.Trans(dp_r, s, dp', s') \wedge p' = \textbf{mnt}(dp', s', p_i, s_i)
\end{aligned}
$$

Can do transition by continuing if unperturbed, otherwise need to find a recovered plan dp_r and do a step of it.

$$
\begin{aligned}
&Final(\textbf{mnt}(dp, s_e, p_i, s_i), s) \equiv \\
&\quad \neg\textbf{ptb}(\textbf{mnt}(dp, s_e, p_i, s_i), s) \wedge Final(dp, s) \vee \\
&\quad \textbf{ptb}(\textbf{mnt}(dp, s_e, p_i, s_i), s) \wedge Do(p_i \| P_{exo}), s_i, s)
\end{aligned}
$$

Can terminate if plan is unperturbed and $Final$ or otherwise if original program p_i can terminate.

Caveat: indefinite postponement.

Related Work: Epistemic Feasibility of Plans in MAS Specifications [Lespérance 01]

Use extended version of ConGolog to specify MAS, e.g.:

$$\begin{array}{l}\mathbf{Subj}(Rob, \exists c\, \textbf{Know}(Rob, comb(Safe1) = c)?; \\ \qquad\qquad dial(Rob, comb(Safe1), Safe1)) \parallel \\ \mathbf{Subj}(Smart, informRef(Smart, Rob, comb(Safe1)))\end{array}$$

Specification is from 3rd person point of view.

Define constructs for capturing subjective execution with and without lookahead.

Formalize epistemic feasibility of plans for agents in MAS. Handles:

- complex (concurrent) programs,
- multiple no-lookahead agents.

Case of multiple agents that do lookahead still open.

Conclusion

New formal account of planning/deliberation with incomplete information; abstract and semantic-based.

Deliberator must produce *epistemically feasible deterministic program*, program for which agent, given initial knowkedge and subsequent sensing, always *knows* what step to take next.

Characterized search in IndiGolog in terms of this notion.

Shown that for certain classes of problems — conformant planning and conditional planning — search for epistemically feasible programs can be limited to programs of a simple syntactic form.

Much earlier work on epistemic feasibility. Ours is first to deal with expressive agent programming language and integrate with transition semantics.

Further Research

Relating 1st and 3rd person accounts of deliberation; compositional development of MAS.

Implementation of search/planning with non-binary sensing actions.

Representing and reasoning with incomplete knowledge.

More general accounts of sensing and knowledge change.

Multiagent planning.

Etc.