Plans, Actions and Dialogues using Linear Logic

Lucas Dixon, Alan Smaill, Tracy Tsang

13 Dec 2007

School of informatics

Overview

A framework to specify and execute agents...

Logical foundation - based on Intuitionistic Linear Logic (ILL)

Agents plan - agents reason by planning (ILL proof)

Agents act and interact - interact through dialogue

React to failure - unexpected things happen, deal with it

Implemented - with case studies in buying and selling hot beverages and reimplementation of Power'79 door problem.

Intuitionistic Linear Logic

Resource sensitive!

Linear Implication: $\texttt{A} \multimap \texttt{B}$ allows resource A to be consumed to produce resource B.

e.g. "$\texttt{eating} : \texttt{hungry} \multimap \texttt{full}$"
eating changes one's state from hungry to full.

Multiplicative Conjunction: $\texttt{A} \otimes \texttt{B}$ indicates that both resources A and B are present.

e.g. "$\texttt{euro} \otimes \texttt{euro} \multimap \texttt{cake}$"
spending 2 euros buys a cake.

Intuitionistic Linear Logic (2)

Additive Disjunction: `A` $\oplus$ `B` indicates that either resource A or B is present, but not both.

e.g. "`lottery : euro` $\multimap$ `winner` $\oplus$ `loser`"
playing a one euro lottery can make you win, or lose, but you cannot choose the outcome!

Additive Conjunction: `A` $\&$ `B` indicates that one can choose, exclusively, to have resource A or resource B.

e.g. "`buy : euro` $\multimap$ `tea` $\&$ `coffee`"
you can choose whether they buy, for one euro, a tea or a coffee.

Intuitionistic Linear Logic (3)

Bang: `!A` indicates that you can get an arbitrary number, including zero, of copies of the resource `A`.

e.g. having "`!(tea ⊸ euro)`"
you can sell tea for one euro as many times as we like (given sufficient tea).

- Deals with Sussman's anomaly in a declarative way:
  if $a_1 \multimap b_1$ and $a_2 \multimap b_2$, then $a_1 \otimes a_2 \multimap b_1 \otimes b_2$
  and the plans can be executed in parallel. Relation to partial order planning?

- Even propositional ILL is undecidable, but it will all be ok...

Provable:	$a \multimap b, b \vdash b$	$!b \vdash b \otimes b$
Unprovable:	$a \otimes a \vdash a$ (extra a)	$a \vdash a \otimes a$ (not enough a)

Agents

Specified as ILL sequents: $\Gamma \vdash C$
almost get a BDI interpretation: Γ = Beliefs + State, and C = Goals, and ...

Proofs are plans: a computational reading ILL proof as a plans.

Reasoning: agents find a plan, which can include the actions of other agents.
(search restricted to depth bound, found shortest-first)

Act: agents then execute the plan.

Failure inspires re-planning: when execution fails, re-plan!

Example: A Lonely Agent Makes a Plan

Agent:

```
    buyfrom A X !: selling A X Y ⊗ have Y ⊸ have X,
     sellingtea !: selling shop tea euro,
  sellingcoffee !: selling shop coffee euro,
             p1 : have euro,
             p2 : have euro
  ⊢ have tea ⊗ have coffee
```

Notation:

```
buyfrom A X !: selling A X Y ⊗ have Y ⊸ have X
= (buyfrom[A,X,]) !: (((selling(A,X,Y)) ⊗ (have(Y))) ⊸ (have(X)))
```

Plan:

```
(buyfrom[shop,tea,euro](sellingtea ⊗ p1))
⊗ (buyfrom[shop,coffee,euro](sellingcoffee ⊗ p2))
```

Expectations as Resources

Agents have expectation of each other, these synchronise interaction:

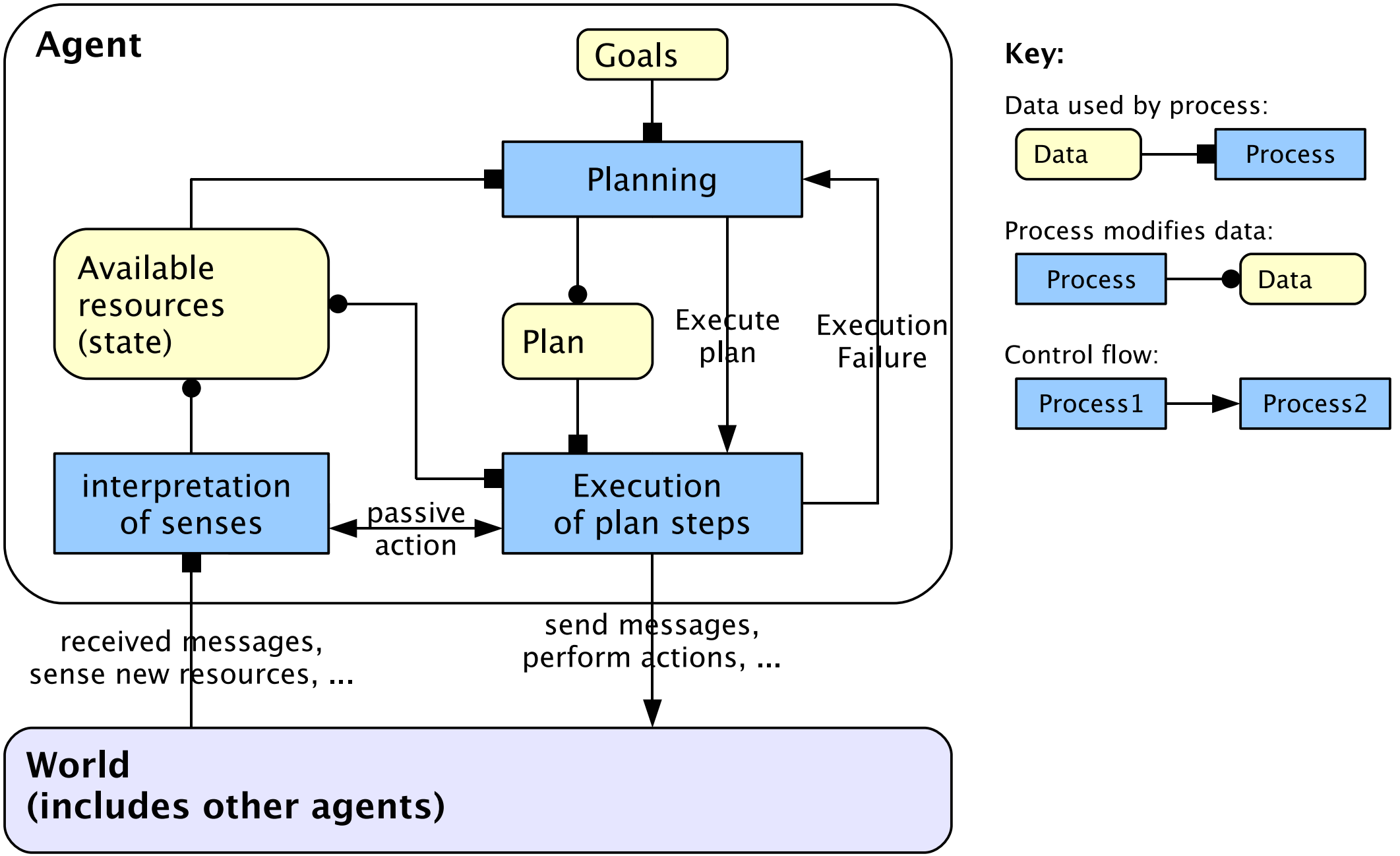
General Ask/Answer:

```
ask A ... !: ... ⊸ iexpectfrom A S ⊗ ...
getanswer A ... !: ... ⊗ iexpectfrom A S ⊸ ...

getasked A ... !: ... ⊸ theyexpect A S ⊗ ...
answer A ... !: ... ⊗ theyexpect A S ⊸ ...
```

Example:

```
ask A (give X) !: has A X ⊸ has A X ⊗ expect A (togiveme X)
getfrom A X !: expect A (togiveme X) ⊗ has A X ⊸ have X
```


Agent
Goals
Planning
Available resources (state)
Plan
Execute plan
Execution Failure
interpretation of senses
passive action
Execution of plan steps
received messages, sense new resources, ...
send messages, perform actions, ...
World (includes other agents)
Key:
Data used by process:
Data
Process
Process modifies data:
Process
Data
Control flow:
Process1
Process2

Execution and Communication

Execution: each plan step has a corresponding execution operator.

Perception: agents interpret predicates from the world.

Initial Speach Act:

```
ask A (fancy X) !: ... ⊸ expect A (fancyitreply X)
```

Plan Operator:

```
getanswer A (fancy X) !: expect A (fancyitreply X) ⊸ know (likes A X)
```

Execution Operator + Interpretation:

```
getanswer A (fancy X) !:
  (says A (ilike X) ⊸ expect A (fancyitreply X) ⊸ know (likes A X))
⊕ (says A (ihate X) ⊸ expect A (fancyitreply X) ⊸ know (hates A X))
```

Power's Domain: Agent Collaboration

- Reimplementation of Power'1979 case study:
- Two locations: **outside** and **inside**.
- One door which can be **open** or **shut**, initially shut.
- Two agents: **John** who is **outside** and **Mary** who in **inside**.
- **John** does not know the effect of moving.
- **Mary** knows but cannot see the state of the world.
- **How to specify agents so they can get John to be inside?**

Solution 1: Dialogue

```
1) John says: How do I achieve know (attr john loc in)?
2) Mary says: what is the loc of john?
3) John says: The loc of john is out
4) Mary says: what is the pos of door
5) John says: The pos of door is shut
6) ** Mary's does: pushes the door open
7) Mary says: OK, do moveto loc in
8) ** John's follows the instruction: do moveto loc in
9) John says: Ok, I've done that.
10) John says: My goal's been achieved! thank you!
```

John's First Plan (the nature of his knowledge)

`do (ask mary (instruction (goal (know (attr john loc in)))))`
$1 \multimap (\mathit{expect\ an\ instruction\ from\ mary})$

`do (getanswer mary (instruction (goal (know (attr john loc in)))))`
$(\mathit{expect\ an\ instruction\ from\ mary}) \multimap (\mathit{expected\ to\ do\ instruction} \otimes \mathit{confirm\ step} \otimes \mathit{confirm\ solved})$

`do (instruction I (goal (know (attr john loc in))))`
$(\mathit{expected\ to\ do\ instruction}) \multimap (\mathit{our\ goal} \otimes \mathit{done\ the\ step})$

`do (answer mary (confirm (step I)))`
$(\mathit{done\ the\ step} \otimes \mathit{confirm\ step}) \multimap 1$

`do (answer mary (confirm (achieved (goal (know (attr john loc in))))))`
$(\mathit{our\ goal} \otimes \mathit{confirm\ solved}) \multimap (\mathit{our\ goal})$

Mary's first plan...

Her first Plan:

`do wait_and_help`

$$1 \multimap (been\ helpful)$$

It goes slightly wrong...

`John: says (ask (instruction (goal (know (attr john loc in)))))`

$$1 \multimap (john\ expects\ instruction)$$

Mary needs to remove the expectation resource... replanning!

Mary's second plan...

John: says (ask (instruction (goal (know (attr john loc in)))))
$1 \multimap (john\ expects\ instruction)$

A new plan...

do (ask john (qattr john loc))
$1 \multimap (expect\ attr\ loc\ answer\ from\ john)$
do (getanswer john (attr john loc in))
$(expect\ attr\ loc\ answer\ from\ john) \multimap (know\ john\ loc\ is\ in)$
do (getanswer john (confirm (achieved
(goal (knows john (attr john loc in))))))
$(know\ john\ loc\ is\ in \otimes john\ expects\ instruction)$
$\multimap (know\ john\ loc\ is\ in \otimes been\ helpful)$

It's all gone wrong again... John's reply... says (attr john loc out)

Mary's third plan...

John: says (attr john loc out)
  ... $\multimap$ *know john is out*

do (pushto door open)
  $1 \multimap$ (*know door is open*)
do (answer john (instruction (movefrom loc out)))
  (*know door is open*) $\multimap$ (*expect confirm step*)
follows john (instruction (movefrom loc out))
  (*know john is out*) $\multimap$ (*know john is in* $\otimes$ *done step*)
do (getanswer john (confirm (step (movefrom loc out))))
  (*done step* $\otimes$ *expect confirm step*) $\multimap$ 1
do (getanswer john (confirm (achieved
        (goal (knows john (attr john loc in))))))
  (... *expectations* $\otimes$ *knowledge* ...) $\multimap$ (*knowledge* ... $\otimes$ *been helpful*)

Solution 2: Dialogue

```
1) John says: How do I achieve know (attr john loc in)?
2) Mary says: what is the loc of john?
3) John says: The loc of john is out
4) Mary says: what is the pos of door
5) John says: The pos of door is shut
7) Mary says: OK, do push door open
8) ** John's follows the instruction: do push door open
9) John says: Ok, I've done that.
10) Mary says: OK, do moveto loc in
11) ** John's follows the instruction: do moveto loc in
12) John says: Ok, I've done that.
13) John says: My goal's been achieved! thankyou!
```

Implementation

- In a Linear Logic programming language (Lolli)
  (Extended with communication channels)
- Ported to Alice (Distributed SML) for distributed execution
- Via meta-interpretation of full ILL (encoded very naive planner in Lolli)
- Declarative (exceptions: one cut to separate execution from planning!
  and communication channels)
- Case studies in this implementation, separate modules for planning/execution.

Summary

An framework for finding and executing plans which include dialogue.

Simple & Admits Reuse: for specifying agents.

Logical Foundation: for both planning, execution, and interaction. Admits formal reasoning.

Supports Graceful Failure: agents operate with imperfect understanding but try to provide sensible behaviour even when it all goes horribly wrong.

Planning to Plan: intentional use of 'imaginary' steps in order to support further planning at execution time.

Summary (2)

Cooperation-Independent: independent of the Gricean Principle of Cooperation: no need for plan recognition and shared goals.

A Rich Language for Specification: agents can make parallel plans, use conditional planning, and plan for the construction of new objects.

Implemented: case studies and implementation platform can be downloaded so you can play with them.

`http://dream.inf.ed.ac.uk/projects/dialoguell/`

Further Work

Formal Reasoning: Formal reasoning about agents and agent interaction

Communication: Ontology matching for communicated predicates

Ontology: Better internal ontology machinery

Learning: Agents that learn - synthesising agents knowledge

Planning: More sophisticated planning (knowledge planners...)

Logic: Issues with negation in ILL

Applications: integration and use in full dialogue systems