Leveraging New Plans in AgentSpeak(PL)

Felipe Meneguzzi
felipe.meneguzzi@kcl.ac.uk
Michael Luck
michael.luck@kcl.ac.uk

[1]Department of Computer Science
King's College London

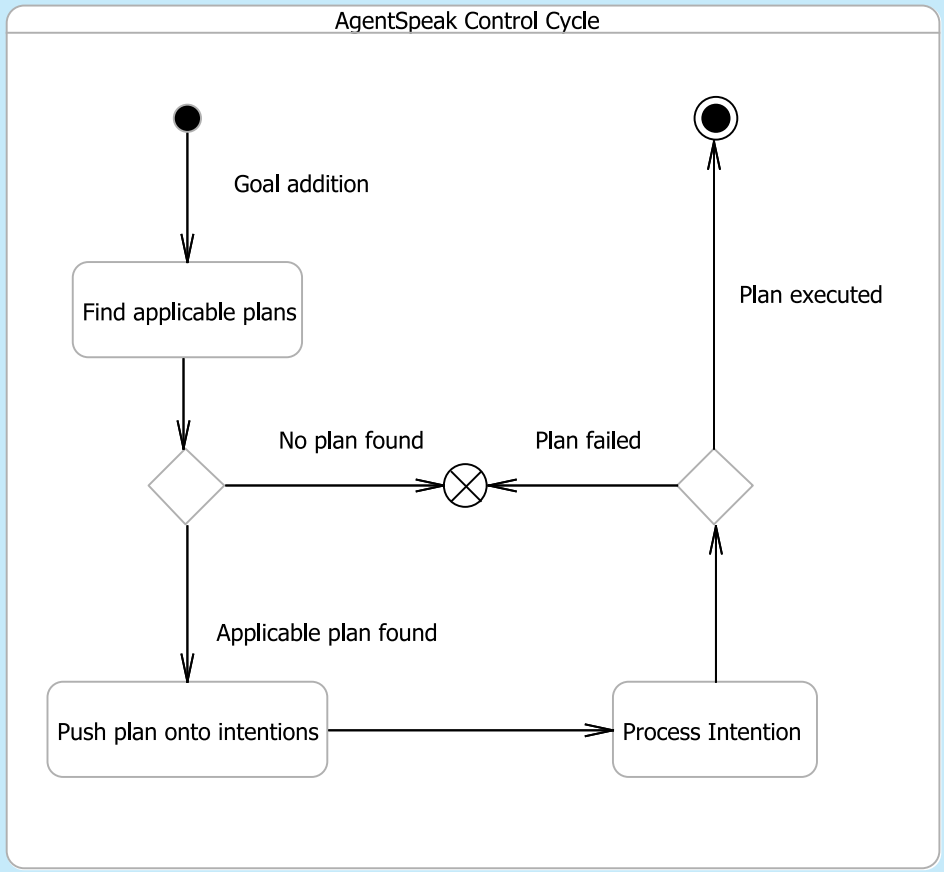
Declarative Agent Languages and Technologies
AAMAS 2008 – Estoril, Portugal

KING'S College LONDON

KING'S College LONDON

AgentSpeak(L)

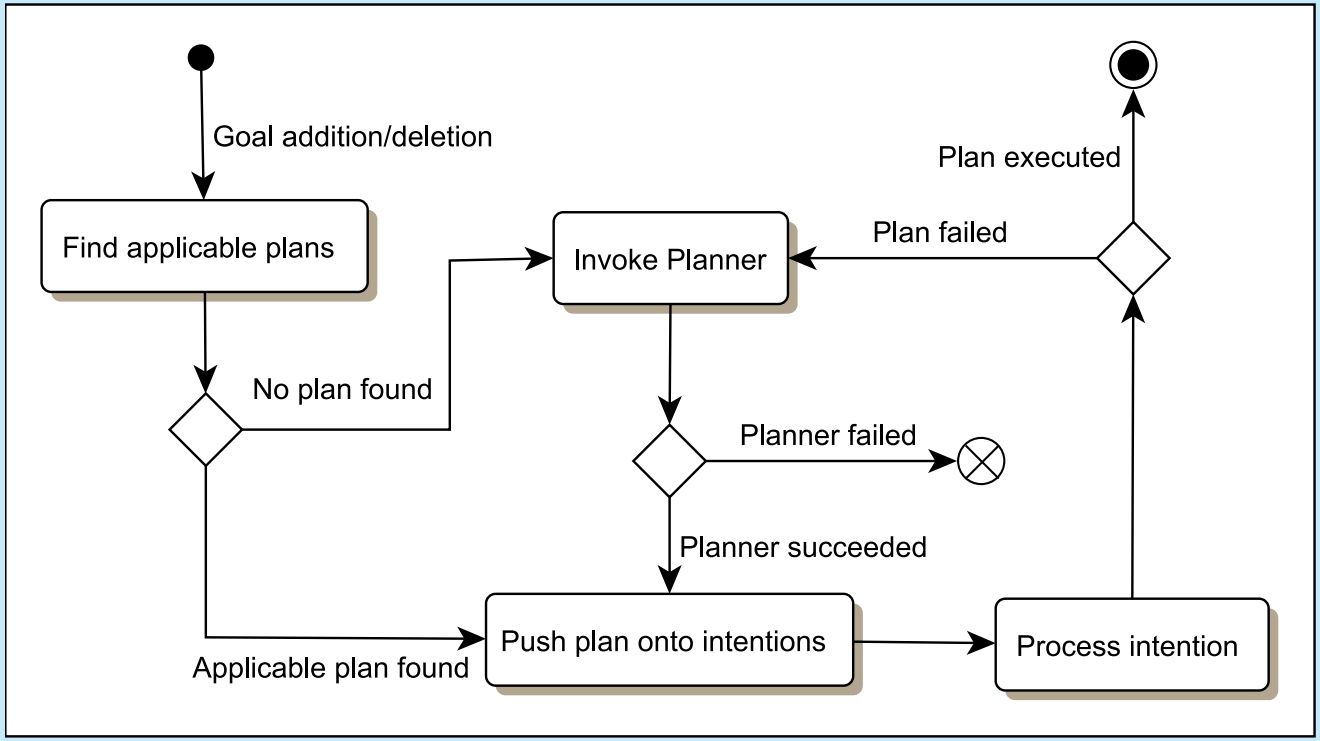
- Procedural agent language
- Based on the BDI model
- Designer specifies plans in a library
  - Plans encode procedures
  - Plans are characterised by trigger and context condition
  - Goals are implicit in the plans



KING'S College LONDON

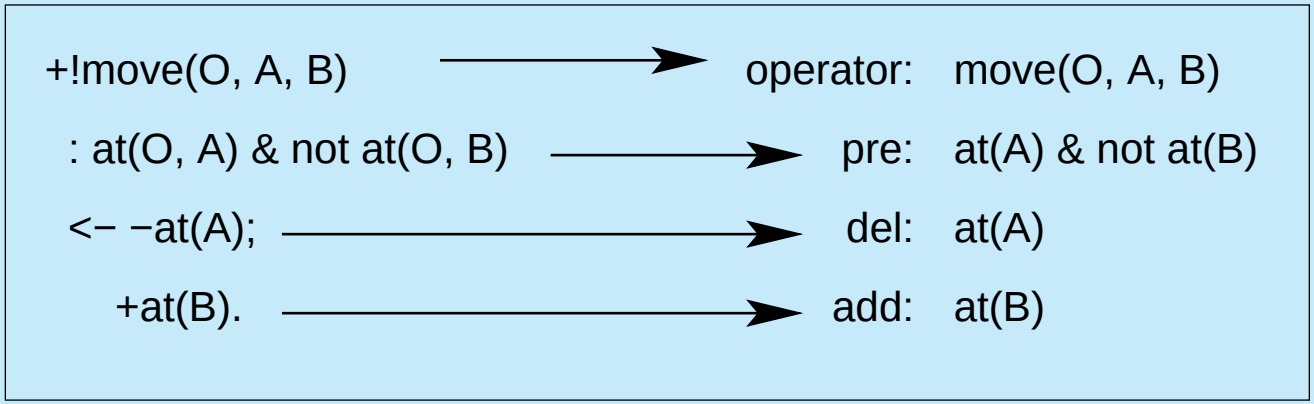
Planning in AgentSpeak(PL)

- AgentSpeak(L) + Planning
  - Standard AgentSpeak(L) language
  - Planner invoked through an atomic action
- In principle, any state-space planner can be used



KING'S College LONDON

AgentSpeak(L) to STRIPS



- Relies on clear similarities between AgentSpeak plans and STRIPS operators
- Desired world state becomes the planners goal
- Belief base becomes the planners start state

KING'S College LONDON

STRIPS to AgentSpeak(L)

STRIPS plan to achieve `battery(full)`:

```
move(1,1)
move(1,2)
charge
```

AgentSpeak(L):

```
+!goal_conj([battery(full)])
:  true
<- !move(1,1);
   !move(1,2);
   !charge.
```

- Each STRIPS action correspond to a low-level AgentSpeak(L) plan
- Plans amount to a series of AgentSpeak(L) subgoals

KING'S College LONDON

Executing Plans and Limitations

- Generated plan is executed as a regular AgentSpeak(L) plan
- Planning is computationally expensive
- New plans should be added to the plan library
- However this is not so trivial:
  - How should a new plan be added?
  - What should the context condition be?

KING'S College LONDON

Leveraging new plans

- Key aspects:
  - Ordering of the plan library - new plans must come before ‘planning plan’
  - Generation of minimum context condition

KING'S College LONDON

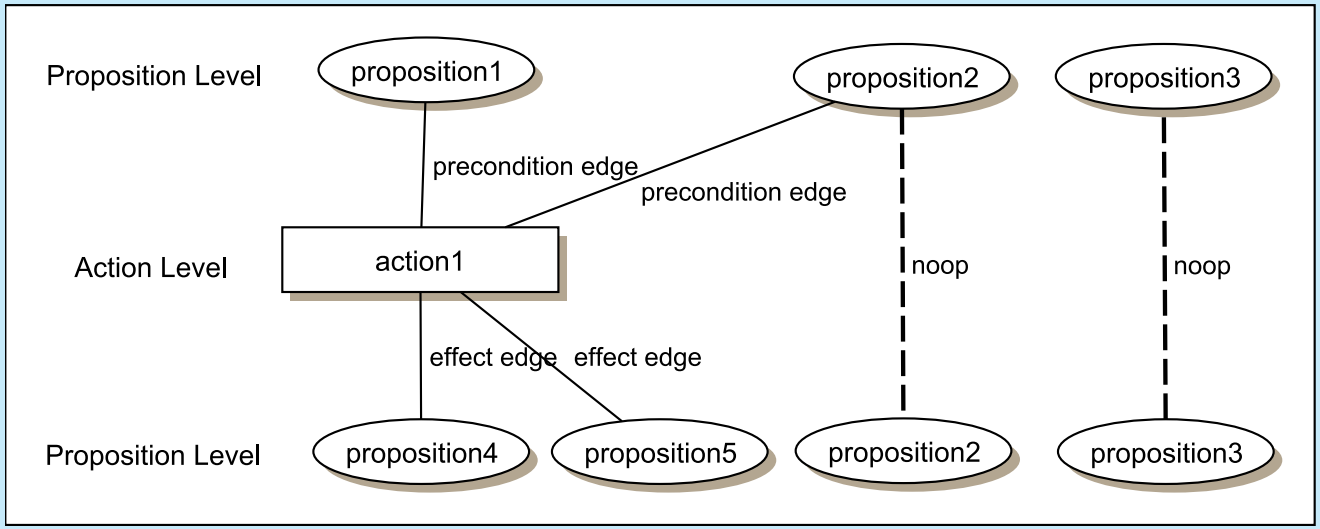
Ordering Example - Pseudo PL

```
+!move(P,A,B) : empty(B) & over(P,A)
   <- ...

+!process(P,A) : over(P,A)
   <- ...

+!goal_conj([over(p1,u1)]) : over(p1,u2)
   <- !move(p1,u2,u1).

// Place new plans here

+!goal_conj(Goals) : true
   <- .plan(Goals).
```

KING'S College LONDON

Planning Graph

- Context generation algorithm uses planning graph
- Directed levelled graph
- Interleaved proposition and action levels
- Preconditions and Effect arcs connect levels



KING'S College LONDON

Generating Context Information

- Preconditions of a plan step must be true earlier in the graph
- Need to propagate preconditions back to previous operators, or to the first level
- Intuitively:
  - Create a planning graph with the target plan
  - Connect preconditions of each action level to the previous one
  - If no action causes the precondition in that level, add a *noop*
  - Propositions propagated to the first level become the context condition

KING'S College LONDON

A Production Cell Example

- Production cell with four processing units and a crane
- Parts can be moved around to processing units
- Processing units can process parts

Operator	Preconditions	Effects

`move(P,A,B)`	`empty(B)`
	`over(P,A)`
		`~empty(B)`
		`~over(P,A)`
		`over(P,B)`
		`empty(A)`

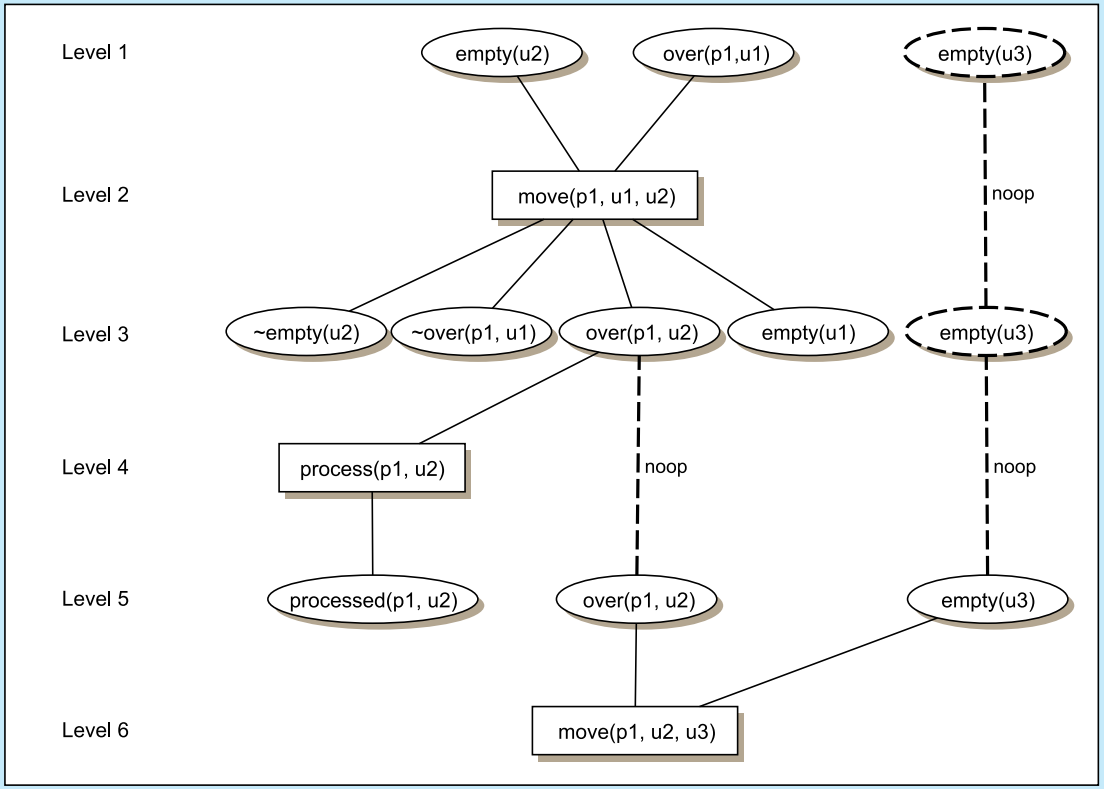
`process(P,A)`	`over(P,A)`	`processed(P)`

KING'S College LONDON

A Production Cell Example: Part II

Generating context condition for plan:

```
move(p1,u1,u2)
process(p1,u2)
move(p1,u2,u3)
```



KING'S College LONDON

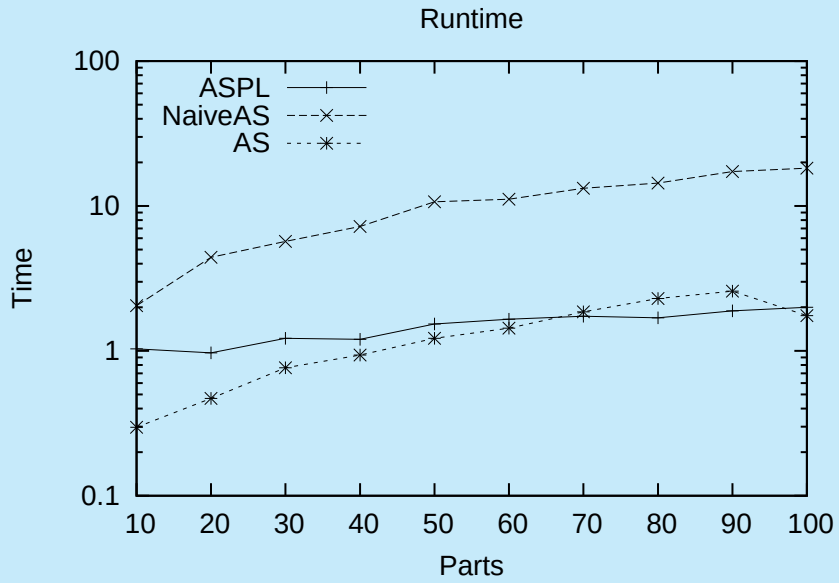
Experiments

- Uses production cell scenario
- Large numbers of parts coming in for processing
- Three types of parts, different sequences of processing
- Measure agent reaction time as new plans are created
- AgentSpeak(L) versus Naive AgentSpeak(PL) versus AgentSpeak(PL)

KING'S College LONDON

Results

- Naive AgentSpeak(PL) very inefficient
- Plan reuse strategy amortises cost of planning
- Over time, computational cost of planning approximates traditional AgentSpeak(L)



KING'S College LONDON

Conclusions and Future Work

- Conclusions:
  - Plan reuse bridges performance gap introduced by planning
  - Algorithm has polynomial complexity (like graph construction in GraphPlan)
  - More complex than necessary, but extensible
- Future Work:
  - Extending algorithm to handle richer operators

KING'S College LONDON

Questions?

KING'S College LONDON

Surprise Slide - Alternative algorithm

- A simpler algorithm may be possible

$Open = \emptyset$
for $i = n$ to 2 **do**
  $Open = Open \cup preconditions(a_i)$
  $Open = Open - postcondition(a_i - 1)$
end for $Open = Open \cup preconditions(a_1)$

KING'S College LONDON