Planning, Execution & Learning
1. Linear & Non-Linear Planning

Reid Simmons

Linear Planning

- Basic Idea
  - *Work on one goal until completely solved before moving on to the next goal*

- Planning Algorithm Maintains Goal ***Stack***

- Implications
  - No interleaving of goal achievement
  - Efficient search if goals do not interact (much)

Means-Ends Analysis

- Basic Idea
  - *Search only relevant aspects of problem*
  - What ***means*** (operators) are available to achieve the desired ***ends*** (goal)

- Find ***difference*** between goal and current state
- Find ***operator*** to reduce difference
- Perform means-ends analysis on new subgoals

GPS

- ***General Problem Solver*** [Newell, Simon, Ernst, 1960's]
  - Introduced concept of means-ends analysis
  - Essentially linear planning using recursive procedure calls as the goal-stack mechanism
- GPS Algorithm *(initial-state, goals)*
  - If *goals* ⊆ *initial-state* then return (*initial-state*, [])
  - **Choose** a difference *d* between *initial-state* and *goals*
  - **Choose** an operator *o* to reduce the difference *d*
  - If no applicable operators, then return (∅, [])
  - (*state*, *plan*) = GPS(*initial-state*, preconditions(*o*))
  - If *state* ≠ ∅ then
    - (*state*, *rest-plan*) = GPS(apply(*o*, *state*), *goals*)
    - *plan* = [*plan*; *o*; *rest-plan*]
  - Return (*state*, *plan*)

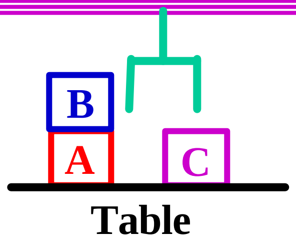
STRIPS

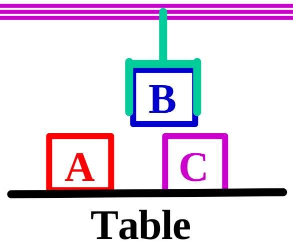
- ***STanford Research Institute Problem Solver*** [Fikes, Nilsson, 1971]
  - Same basic idea as GPS, *but*
  - Solved the frame problem (“STRIPS assumption”)
  - Introduced operator representation
  - Operationalized notion of ***difference***, ***subgoals***, and operator ***application***
  - Dealt (somewhat) with plan execution and learning

STRIPS Algorithm

- STRIPS (*initial-state, goals*)
  - *state* = *initial-state*; *plan* = []; *stack* = []
  - Push *goals* on *stack*
  - Repeat until *stack* is empty
    - If top of *stack* is **goal** that matches *state*, then pop *stack*
    - Else if top of stack is a **conjunctive goal** *g*, then
      - **Select** an ordering for the subgoals of *g*, and push them on *stack*
    - Else if top of *stack* is a **simple goal** *sg*, then
      - **Choose** an operator *o* whose add-list matches goal *sg*
      - Replace goal *sg* with operator *o*
      - Push the preconditions of *o* on the *stack*
    - Else if top of *stack* is an **operator** *o*, then
      - *state* = apply(*o*, *state*)
      - *plan* = [*plan*; *o*]

STRIPS Meets the Blocks World





Pickup_from_table(b)
Pre: Block(b), Handempty
Clear(b), On(b, Table)
Add: Holding(b)
Delete: Handempty,
On(b, Table)

Putdown_on_table(b)
Pre: Block(b), Holding(b)
Add: Handempty,
On(b, Table)
Delete: Holding(b)

Pickup_from_block(b, c)
Pre: Block(b), Handempty
Clear(b), On(b, c), Block(c)
Add: Holding(b), Clear(c)
Delete: Handempty,
On(b, c)

Putdown_on_block(b, c)
Pre: Block(b), Holding(b)
Block(c), Clear(c), b 1 c
Add: Handempty, On(b, c)
Delete: Holding(b), Clear(c)

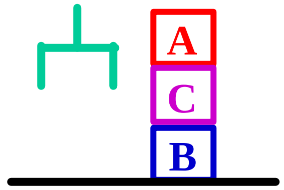
STRIPS Blocks-World Example

1. Stack

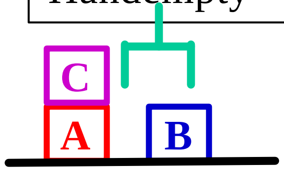
On(A, C) On(C, B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty



Goal



Initial State

2. Stack

On(A, C) On(C, B)

On(A, C)

On(C, B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

3. Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

Holding(C) Clear(B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

4. Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

Holding(C) Clear(B)

Holding(C)

Clear(B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

STRIPS Blocks-World Example

5. Stack

On(A, C) On(C, B)
On(A, C)
Put_Block(C, B)
Holding(C) Clear(B)
Holding(C)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

6. Stack

On(A, C) On(C, B)
On(A, C)
Put_Block(C, B)
Holding(C) Clear(B)
Pick_Block(C)
Handempty Clear(C) On(C, ?b)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

7. Stack

On(A, C) On(C, B)
On(A, C)
Put_Block(C, B)
Holding(C) Clear(B)
Pick_Block(C)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

8. Stack

On(A, C) On(C, B)
On(A, C)
Put_Block(C, B)
Holding(C) Clear(B)

[Pick(C)]

State

Clear(B)
Clear(C)
On(A, Table)
On(B, Table)
Holding(C)
Clear(A)

STRIPS Blocks-World Example

9. Stack

On(A, C) On(C, B)

On(A, C)

Put_Block(C, B)

State

Clear(B)
Clear(C)
On(A, Table)
On(B, Table)
Holding(C)
Clear(A)

[Pick(C)]

10. Stack

On(A, C) On(C, B)

On(A, C)

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick(C); Put(C, B)]

11. Stack

On(A, C) On(C, B)

Put_Block(A, C)

Holding(A) Clear(C)

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick(C); Put(C, B)]

12. Stack

On(A, C) On(C, B)

Put_Block(A, C)

Holding(A) Clear(C)

Holding(A)

Clear(C)

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick(C); Put(C, B)]

STRIPS Blocks-World Example

13. **Stack**

On(A, C) On(C, B)
Put_Block(A, C)
Holding(A) Clear(C)
Holding(A)

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick(C); Put(C, B)]

14. **Stack**

On(A, C) On(C, B)
Put_Block(A, C)
Holding(A) Clear(C)
Pick_Table(A)
Handempty Clear(A) On(A, Table)

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick(C); Put(C, B)]

15. **Stack**

On(A, C) On(C, B)
Put_Block(A, C)
Holding(A) Clear(C)
Pick_Table(A)

State

Clear(C)
On(A, Table)
On(B, Table)
Clear(A)
Handempty
On(C, B)

[Pick(C); Put(C, B)]

16. **Stack**

On(A, C) On(C, B)
Put_Block(A, C)
Holding(A) Clear(C)

State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

[Pick(C); Put(C, B); PickT(A)]

STRIPS Blocks-World Example

17. Stack

On(A, C) On(C, B)

Put_Block(A, C)

State

Clear(C)
On(B, Table)
Clear(A)
On(C, B)
Holding(A)

[Pick(C); Put(C, B); PickT(A)]

18. Stack

On(A, C) On(C, B)

State

On(B, Table)
Clear(A)
On(C, B)
Handempty
On(A, C)

[Pick(C); Put(C, B); PickT(A); Put(A, C)]

19. Stack

State

On(B, Table)
Clear(A)
On(C, B)
Handempty
On(A, C)

[Pick(C); Put(C, B); PickT(A); Put(A, C)]

Properties of Planning Algorithms

- **Soundness**
  - A planning algorithm is ***sound*** if all solutions found are legal plans
    - All preconditions and goals are satisfied
    - No constraints are violated (temporal, variable binding)
- **Completeness**
  - A planning algorithm is ***complete*** if a solution can be found whenever one actually exists
  - A planning algorithm is ***strictly complete*** if all solutions are included in the search space
- **Optimality**
  - A planning algorithm is ***optimal*** if the order in which solutions are found is consistent with some measure of plan quality

Linear Planning: Discussion

- **Advantages**
  - Reduced search space, since goals are solved one at a time
  - Advantageous if goals are (mainly) independent
  - Linear planning is ***sound***

- **Disadvantages**
  - Linear planning may produce ***suboptimal*** solutions (based on the number of operators in the plan)
  - Linear planning is ***incomplete***

Suboptimal Plans

- Result of linearity, ***goal interactions*** and poor ***goal ordering***

Load(o, p, loc)
  Pre: At(o, loc), At(p, loc)
  Add: Inside(o, p)
  Delete: At(o, loc)

Unload(o, p, loc)
  Pre: Inside(o, p), At(p, loc)
  Add: At(o, loc)
  Delete: Inside(o, p)

Fly(p, from, to)
  Pre: At(p, from)
  Add: At(p, to)
  Delete: At(p, from)

- Initial State: At(Obj1, LocA), At(Obj2, LocA), At(747, LocA)
- Goals: At(Obj1, LocB), At(Obj2, LocB)
- Plan: [**Load(Obj1, 747, LocA); Fly(747, LocA, LocB);
  Unload(Obj1, 747, LocB); Fly(747, LocB, LocA);
  Load(Obj2, 747, LocA); Fly(747, LocA, LocB); Unload(Obj2, 747, LocB)]**

STRIPS and the FedEx World

1.

Stack	**State**
At(Obj1, LocB)	At(Obj1, LocA)
At(Obj2, LocB)	At(Obj2, LocA)
	At(747, LocA)

2.

Stack	**State**
At(Obj1, LocB) At(Obj2, LocB)	At(Obj1, LocA)
At(Obj2, LocB)	At(Obj2, LocA)
At(Obj1, LocB)	At(747, LocA)

3.

Stack	**State**
At(Obj1, LocB) At(Obj2, LocB)	At(Obj1, LocA)
At(Obj2, LocB)	At(Obj2, LocA)
Unload(Obj1, 747, LocB)	At(747, LocA)
Inside(Obj1, 747) At(747, LocB)	

4.

Stack	**State**
At(Obj1, LocB) At(Obj2, LocB)	At(Obj1, LocA)
At(Obj2, LocB)	At(Obj2, LocA)
Unload(Obj1, 747, LocB)	At(747, LocA)
Inside(Obj1, 747) At(747, LocB)	
Inside(Obj1, 747)	
At(747, LocB)	

STRIPS and the FedEx World

5. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Inside(Obj1, 747)
Fly(747, ?loc, LocB)
At(747, ?loc)

State

At(Obj1, LocA)
At(Obj2, LocA)
At(747, LocA)

6. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Inside(Obj1, 747)
Fly(747, LocA, LocB)

State

At(Obj1, LocA)
At(Obj2, LocA)
At(747, LocA)

7. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Inside(Obj1, 747)

[Fly(747, LocA, LocB)]

State

At(Obj1, LocA)
At(Obj2, LocA)
At(747, LocB)

8. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Load(Obj1, 747, ?loc)
At(Obj1, ?loc) At(747, ?loc)

[Fly(747, LocA, LocB)]

State

At(Obj1, LocA)
At(Obj2, LocA)
At(747, LocB)

STRIPS and the FedEx World

9. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Load(Obj1, 747, ?loc) **[Fly(747, LocA, LocB)]**
At(Obj1, ?loc) At(747, ?loc)
At(747, ?loc)
At(Obj1, ?loc)

State

At(Obj1, LocA)
At(Obj2, LocA)
At(747, LocA)

10. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Load(Obj1, 747, LocA) **[Fly(747, LocA, LocB)]**
At(Obj1, LocA) At(747, LocA)
At(747, LocA)

State

At(Obj1, LocA)
At(Obj2, LocA)
At(747, LocB)

STRIPS and the FedEx World

11. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Load(Obj1, 747, LocA)
At(Obj1, LocA) At(747, LocA)
Fly(747, ?loc, LocA)
At(747, ?loc)

State

At(Obj1, LocA)
At(Obj2, LocA)
At(747, LocB)

[Fly(747, LocA, LocB)]

12. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Load(Obj1, 747, LocA)
At(Obj1, LocA) At(747, LocA)
Fly(747, LocB, LocA)

State

At(Obj1, LocA)
At(Obj2, LocA)
At(747, LocB)

[Fly(747, LocA, LocB)]

13. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)

State

At(Obj2, LocA)
At(747, LocA)
Inside(Obj1, 747)

[Fly(747, LocA, LocB);
Fly(747, LocB, LocA);
Load(Obj1, 747, LocA)]

STRIPS and the FedEx World

14. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
At(747, LocB)
Inside(Obj1, 747)

State

At(Obj2, LocA)
At(747, LocA)
Inside(Obj1, 747)

[Fly(747, LocA, LocB);
Fly(747, LocB, LocA);
Load(Obj1, 747, LocA)]

15. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)
Fly(747, ?loc, LocB)
At(747, ?loc)

State

At(Obj2, LocA)
At(747, LocA)
Inside(Obj1, 747)

[Fly(747, LocA, LocB);
Fly(747, LocB, LocA);
Load(Obj1, 747, LocA)]

16. Stack

At(Obj1, LocB) At(Obj2, LocB)
At(Obj2, LocB)
Unload(Obj1, 747, LocB)
Inside(Obj1, 747) At(747, LocB)

State

At(Obj2, LocA)
At(747, LocB)
Inside(Obj1, 747)

[Fly(747, LocA, LocB);
Fly(747, LocB, LocA);
Load(Obj1, 747, LocA);
Fly(747, LocA, LocB)]

The “Sussman Anomaly”

1. Stack

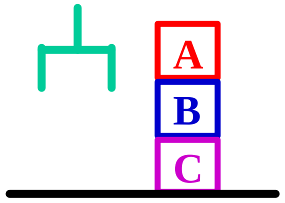
On(A, B) On(B, C)

On(A, B)

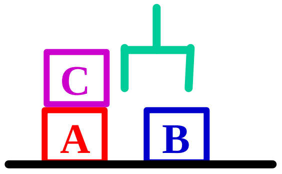
On(B, C)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty



Goal



Initial State

2. Stack

On(A, B) On(B, C)

On(B, C)

Put_Block(A, B)

Holding(A) Clear(B)

Holding(A)

Clear(B)

State

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty

The “Sussman Anomaly”

3.

Stack	State
On(A, B) On(B, C)	Clear(B)
On(B, C)	Clear(C)
Put_Block(A, B)	On(C, A)
Holding(A) Clear(B)	On(A, Table)
Holding(A)	On(B, Table)
Pick_Table(A)	Handempty
Handempty Clear(A) On(A, Table)	
Clear(A)	
Handempty	
On(A, Table)	

4.

Stack	State
On(A, B) On(B, C)	Clear(B)
On(B, C)	Clear(C)
Put_Block(A, B)	On(C, A)
Holding(A) Clear(B)	On(A, Table)
Holding(A)	On(B, Table)
Pick_Table(A)	Handempty
Handempty Clear(A) On(A, Table)	
Pick_Block(C, A)	
Handempty Clear(C) On(C, A)	

The "Sussman Anomaly"

5. Stack

On(A, B) On(B, C)
On(B, C)
Put_Block(A, B)
Holding(A) Clear(B)
Holding(A)
Pick_Table(A)
Handempty Clear(A) On(A, Table)

[Pick(C,A)]

State

Clear(B)
Clear(C)
On(A, Table)
On(B, Table)
Holding(C)

6. Stack

On(A, B) On(B, C)
On(B, C)
Put_Block(A, B)
Holding(A) Clear(B)
Holding(A)
Pick_Table(A)
Handempty Clear(A) On(A, Table)
Put_Table(C)
Holding(C)

[Pick(C,A)]

State

Clear(B)
Clear(C)
On(A, Table)
On(B, Table)
Holding(C)

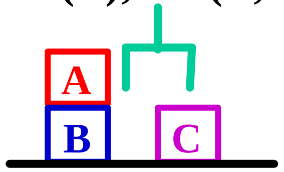
The "Sussman Anomaly"

7. Stack

On(A, B) On(B, C)

On(B, C)

[Pick(C,A); PutT(C);
PickT(A); Put(A, B)]



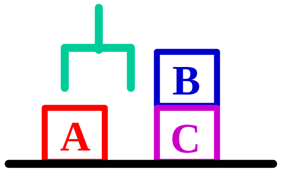
State

Clear(C)
On(B, Table)
On(C, Table)
Clear(A)
On(A, B)
Handempty

8. Stack

On(A, B) On(B, C)

[Pick(C,A); PutT(C);
PickT(A); Put(A, B);
Pick(A, B); PutT(A);
PickT(B); Put(B, C)]



State

On(C, Table)
Clear(B)
Clear(A)
On(A, Table)
On(B, C)
Handempty

9. Stack

On(A, B) On(B, C)

On(A, B)

On(B, C)

[Pick(C,A); PutT(C);
PickT(A); Put(A, B);
Pick(A, B); PutT(A);
PickT(B); Put(B, C)]

State

On(C, Table)
Clear(B)
Clear(A)
On(A, Table)
On(B, C)
Handempty

10. Stack

On(A, B) On(B, C)

[Pick(C,A); PutT(C);
PickT(A); Put(A, B);
Pick(A, B); PutT(A);
PickT(B); Put(B, C);
PickT(A); Put(A, B)]

State

On(C, Table)
Clear(A)
On(B, C)
On(A, B)
Handempty

Unsolvable Problems

- Result of linearity and poor ***irreversible actions***

```
Load(o, p, loc)                      Unload(o, p, loc)
  Pre: At(o, loc), At(p, loc)          Pre: Inside(o, p), At(p, loc)
  Add: Inside(o, p)                    Add: At(o, loc)
  Delete: At(o, loc)                   Delete: Inside(o, p)
```

Fly(p, from, to)
  Pre: At(p, from), **Have-Fuel(p)**
  Add: At(p, to)
  Delete: At(p, from), **Have-Fuel(p)**

- Initial State: At(Obj1, LocA), At(Obj2, LocA), At(747, LocA), Have-Fuel(747)
- Goals: At(Obj1, LocB), At(Obj2, LocB)

STRIPS and the UPS World

- I: Try Achieving Goal ***At(Obj1, LocB)*** First
  - **[Load(Obj1, LocA, 747); Fly(747, LocA, LocB); Unload(Obj1, 747, LocB)]**
  - But, now cannot achieve **Fly(747, LocB, LocA)**, since no fuel!
- II: Try Achieving Goal ***At(Obj2, LocB)*** First
  - **[Load(Obj2, LocA, 747); Fly(747, LocA, LocB); Unload(Obj2, 747, LocB)]**
  - But, now cannot achieve **Fly(747, LocB, LocA)**, since no fuel!
- ***Either way, the problem is unsolvable by STRIPS***

Non-Linear Planning

- Basic Idea
  - Use goal ***set*** instead of goal stack
  - Include in the search space all possible subgoal orderings
    - Handles goal interactions by ***interleaving***
- **Advantages**
  - Non-linear planning is ***sound***
  - Non-linear planning is ***complete***
  - Non-linear planning may be ***optimal*** with respect to plan length (depending on search strategy employed)
- **Disadvantages**
  - Larger search space, since all possible goal orderings may have to be considered
  - Somewhat more complex algorithm; More bookkeeping

Non-Linear Planning Algorithm

```
• NLP (initial-state, goals)
  – state = initial-state;  plan = []; goalset = goals; opstack = []
  – Repeat until goalset is empty
      • Choose a goal g from the goalset
      • If g does not match state, then
          – Choose an operator o whose add-list matches goal g
          – Push o on the opstack
          – Add the preconditions of o to the goalset
      • While all preconditions of operator on top of opstack
        are met in state
          – Pop operator o from top of opstack
          – state = apply(o, state)
          – plan = [plan; o]
```

Non-Linear FedEx World

	Goals	State	Ops	Plan
1.	At(Obj1, LocB)	At(Obj1, LocA)		[]
	At(Obj2, LocB)	At(Obj2, LocA)		
		At(747, LocA)		
2.	At(Obj1, LocB)	At(Obj1, LocA)	Unload(Obj2, 747, LocB)	[]
	Inside(Obj2, 747)	At(Obj2, LocA)		
	At(747, LocB)	At(747, LocA)		
3.	At(Obj1, LocB)	At(Obj1, LocA)	Load(Obj2, 747, ?loc1)	[]
	At(747, LocB)	At(Obj2, LocA)	Unload(Obj2, 747, LocB)	
	At(747, ?loc1)	At(747, LocA)		
	At(Obj1, ?loc1)			
4.	At(Obj1, LocB)	At(Obj1, LocA)	Unload(Obj2, 747, LocB)	[Load(Obj2, 747, LocA)]
	At(747, LocB)	At(747, LocA)		
		Inside(Obj2, 747)		

Non-Linear FedEx World

	Goals	State	Ops	Plan

5.	***At(Obj1, LocB)***
	At(747, LocB)
		At(Obj1, LocA)
		At(747, LocA)
		Inside(Obj2, 747)
			Unload(Obj2, 747, LocB)	[Load(Obj2, 747, LocA)]

6.	At(747, LocB)
	Inside(Obj1, 747)
		At(Obj1, LocA)
		At(747, LocA)
		Inside(Obj2, 747)
			Unload(Obj1, 747, LocB)
			Unload(Obj2, 747, LocB)
				[Load(Obj2, 747, LocA)]

7.	At(747, LocB)
	At(747, ?loc2)
	At(Obj1, ?loc2)
		At(Obj1, LocA)
		At(747, LocA)
		Inside(Obj2, 747)
			Load(Obj1, 747, ?loc2)
			Unload(Obj1, 747, LocB)
			Unload(Obj2, 747, LocB)
				[Load(Obj2, 747, LocA)]

8.	***At(747, LocB)***	At(747, LocA)
		Inside(Obj2, 747)
		Inside(Obj1, 747)
			Unload(Obj1, 747, LocB)
			Unload(Obj2, 747, LocB)
				[Load(Obj2, 747, LocA);
				Load(Obj1, 747, LocA)]

Non-Linear FedEx World

	Goals	State	Ops	Plan

9.	***At(747, LocA)***	At(747, LocA)
		Inside(Obj2, 747)
		Inside(Obj1, 747)
			Fly(747, LocA, LocB)
			Unload(Obj1, 747, LocB)
			Unload(Obj2, 747, LocB)
				[Load(Obj2, 747, LocA);
				Load(Obj1, 747, LocA)]

10.		At(747, LocB)
		Inside(Obj2, 747)
		Inside(Obj1, 747)
			Unload(Obj1, 747, LocB)
			Unload(Obj2, 747, LocB)
				[Load(Obj2, 747, LocA);
				Load(Obj1, 747, LocA);
				Fly(747, LocA, LocB)]

11.		At(747, LocB)
		At(Obj2, LocB)
		At(Obj1, LocB)
				[Load(Obj2, 747, LocA);
				Load(Obj1, 747, LocA);
				Fly(747, LocA, LocB);
				Unload(Obj1, 747, LocB);
				Unload(Obj2, 747, LocB)]