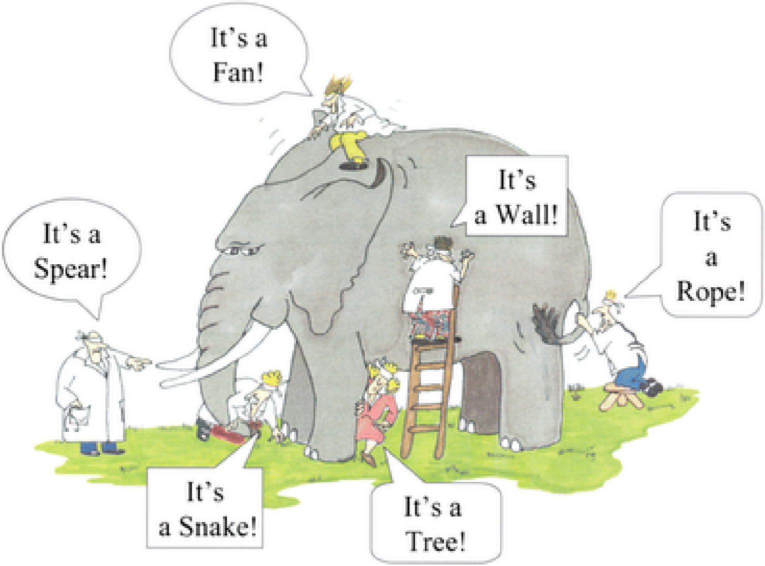
Lifting Delete Relaxation Heuristics To Successor Generator Planning

Michael Katz[1] Dany Moshkovich[1] Erez Karpas[2]

[1]IBM Haifa Research Lab
[2]Technion — Israel Institute of Technology

Workshop on Heuristics and Search for Domain-Independent Planning — ICAPS 2016

What is Planning?



What is Planning?

- Given initial state, goal, and description of possible actions, find a sequence of actions which leads from the initial state to the goal
- How are these described?
  - Symbolically, e.g., in PDDL
  - With a black-box successor generator

What is Planning?

- Given initial state, goal, and description of possible actions, find a sequence of actions which leads from the initial state to the goal
- How are these described?
    - Symbolically, e.g., in PDDL
    - With a black-box successor generator

The Symbolic Approach

- Problem description is given by:
  - A set of *state variables* F
  - A set of *actions* A
    - Preconditions and effects are partial assignments to F
  - Initial state is a complete assignment to F
  - Goal is a partial assignment to F
- PDDL also adds types, objects, parameters

The Black-Box Approach

- Problem description is given by:
    - Initial state s_0 — some opaque object
    - Successor generator *succ* — a function which can be applied to a state *s* and returns a list of applicable actions and their successor
    - Goal test *goal*? — a function which can be applied to a state and returns whether that state is a goal state

Black-Box vs. Symbolic Approach

- Symbolic approach allows us to automatically derive *heuristics*
- Symbolic approach is sometimes harder to understand
    - I realize this is the wrong room to make this point, but consider students you've taught PDDL
- Black box approach allows for more efficient successor generation
    - No need for "unnecessary" parameters in actions, e.g., in DRIVE(?FROM, ?TO) — we know where we are (see also Areces, Bustos, Dominguez and Hoffmann, 2014)
    - No need for static predicates which encode math, e.g., NEXT(N1,N2), NEXT(N2,N3), ...

Black-Box vs. Symbolic Approach

- Symbolic approach allows us to automatically derive *heuristics*
- Symbolic approach is sometimes harder to understand
  - I realize this is the wrong room to make this point, but consider students you've taught PDDL
- Black box approach allows for more efficient successor generation
  - No need for "unnecessary" parameters in actions, e.g., in DRIVE(?FROM, ?TO) — we know where we are (see also Areces, Bustos, Dominguez and Hoffmann, 2014)
  - No need for static predicates which encode math, e.g., NEXT(N1,N2), NEXT(N2,N3), . . .

Black-Box vs. Symbolic Approach

- Symbolic approach allows us to automatically derive *heuristics*
- Symbolic approach is sometimes harder to understand
  - I realize this is the wrong room to make this point, but consider students you've taught PDDL
- Black box approach allows for more efficient successor generation
  - No need for "unnecessary" parameters in actions, e.g., in DRIVE(?FROM, ?TO) — we know where we are (see also Areces, Bustos, Dominguez and Hoffmann, 2014)
  - No need for static predicates which encode math, e.g., NEXT(N1,N2), NEXT(N2,N3), . . .

The Best of Both Worlds?

- We introduce the *Semi Black-Box* approach
- Combines the flexibility of the black-box approach with the ability to automatically derive a heuristic
- Implemented in a Java framework called *object oriented planning*

Object Oriented Planning

- A state consists of a set of *entities*
  - These are objects in the *pddl* sense, which are implemented as objects in Java
  - Each entity has an internal state (e.g,. location)
- Operators (ungrounded actions) are implemented as Java classes
  - Operator implements two functions: ISAPPLICABLE and APPLY
  - Successor generator calls ISAPPLICABLE for all possible parameter combinations (according to entities in given state)
  - If an operator + parameters is applicable, APPLY is called to generate the successor
- So far, this is purely a black-box framework

Object Oriented Planning: Archetypes

- The framework also contains a set of entities and operators with *known behavior*
  - Intuitively, a set of entities and operators that corresponds to a known PDDL domain
- We refer to these known entities and operators as *archetypes*
  - Example: MobileEntity and Move operator
- Users can *inherit* from these, and modify their behavior
- Note: unlike PDDL, actions can create or destroy entities

Object Oriented Planning: Archetypes

- The framework also contains a set of entities and operators with *known behavior*
  - Intuitively, a set of entities and operators that corresponds to a known PDDL domain
- We refer to these known entities and operators as *archetypes*
  - Example: MobileEntity and Move operator
- Users can *inherit* from these, and modify their behavior

Example: Vehicle Entity

```
public class Vehicle extends MobileEntity {
  private int vehicleCurrentCapacity;
  private final int maximalCapacity;
  public Vehicle(String entityId, long time,
      long timeBound, Place location,
      RoutingRequest constraints,
      int maxCapacity) {
    super(entityId, time, timeBound,
                    location, constraints);
    maximalCapacity = maxCapacity;
    vehicleCurrentCapacity = -1;
  }
}
```

Example: Drive Action

```
public class Drive extends Move {
  public Drive(
      ILocationService locationService){
    super("DRIVE_ACTION", Vehicle.class,
          Place.class, locationService,
          1, 0, 0);
  }

  @Override
  public boolean isApplicable(IState state,
        IEntity[] params) {
    Vehicle v = (Vehicle)params[0];
    return
      (v.getVehicleCurrentCapacity()>-1)
      && super.isApplicable(state,params);
  }
```

Heuristics for Object Oriented Planning Problems

- High level idea
  - Project the problem onto the known aspects
  - Use a known heuristic on this symbolic planning problem
  - Can not make any guarantees on admissibilty or dead-end safety because we can not know what behavior the user overrode
- However, because entities can be created and destroyed, we can not ground the problem during preprocessing
- We implemented a lifted variant of $h_{FF}(\Pi^C)$
  - Actions are grounded on-the-fly, as they are added to the relaxed planning graph
  - Since we know the domain, we can also choose a good C

Empirical Evaluation

- We tested on two domains:
- Commuter car pooling
  - A set of commuters can drive vehicles and pick each other up
  - Each commuter has temporal constraints on when they can leave their house, get to work, leave work, and get back home
  - Supported by another feature of our framework — temporally expressive goals
  - Compared to LAMA and GBFS with FF heuristic (running on PDDL encoding), to our framework without the heuristic, and to optic on PDDL 2.1 encoding
- Evolution
  - A set of entities can move between different locations and mate (if they are in the same place)
  - Goal is to create an entity whose ancestors include a given set of entities
  - Impossible to model in PDDL, so only SBB was tested

Evaluation: Commuter Car Pooling — Cost and Quality

	Plan cost				Quality			
	LAMA	FF	SBB	BB	LAMA	FF	SBB	BB
02_0	1108	1108	1108	1108	1.00	1.00	1.00	1.00
02_1	1108	1108	1108	1108	1.00	1.00	1.00	1.00
04_0	2176		1136		0.52	0.00	1.00	0.00
04_1	1136	1136	1136		1.00	1.00	1.00	0.00
04_2	1140	1140	1140	1140	1.00	1.00	1.00	1.00
04_3	1136	1136	1136	1136	1.00	1.00	1.00	1.00
06_0	5324		4332		0.81	0.00	1.00	0.00
06_1	5364		5374		1.00	0.00	1.00	0.00
06_2	4304		3270		0.76	0.00	1.00	0.00
06_3	3264	3304	2244		0.69	0.68	1.00	0.00
06_4	2244	2244	2244	2244	1.00	1.00	1.00	1.00
08_0	7472		5426		0.73	0.00	1.00	0.00
08_1	6412		6402		1.00	0.00	1.00	0.00
08_6			∞		0.00	0.00	1.00	0.00
10_0	9560				1.00	0.00	0.00	0.00
10_2	8560				1.00	0.00	0.00	0.00
Sum					13.51	6.68	14.00	5.00

Evaluation: Commuter Car Pooling — Planning Time

	Total time				
	optic	LAMA	FF	SBB	BB
02_0	0.1	0.2	0.1	0.1	0.1
02_1	0.0	0.1	0.1	0.1	0.0
04_0	8.3	1322.5		280.4	
04_1	0.3	1203.7	737.8	29.3	
04_2	22.2	93.2	25.9	1.5	2.0
04_3	0.2	4.8	1.1	0.3	0.6
06_0		50.3		3.5	
06_1		22.2		5.3	
06_2	437.4	35.1		1.1	
06_3		22.6	627.0	471.7	
06_4		54.7	304.6	6.7	25.9
08_0		151.3		39.0	
08_1		208.6		6.7	
08_6				239.9	
10_0		244.6			
10_2		368.4			

Evaluation: Evolution

	cost	time
03_3	250	0.564
04_3	250	0.311
04_4	340	0.511
06_3	240	1.27
06_4	320	1.224
06_5		
08_3	220	0.482
08_4	320	6.88
08_5		
08_6		
10_3	220	2.368
10_4		
10_5		
10_6		
10_7		

Summary

- Presented the object oriented planning framework, which adds some symbolic elements to black-box planning
- Empirically showed the benefits of this approach
- Future work
    - Add more archetypes to the framework, possibly with a generic mechanism for adding PDDL notations
    - Empirical evaluation of people's ease-of-use of BB, SBB, and PDDL

Thank You

Questions?