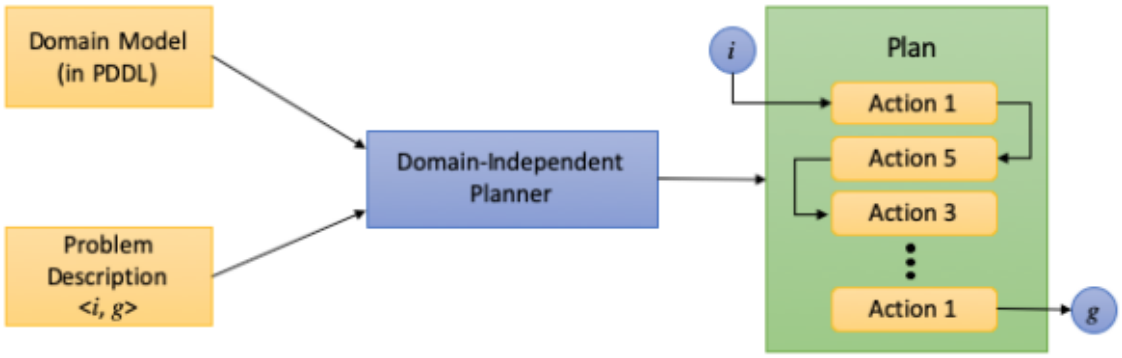
Classical Planning

Planning Domain Definition Language (PDDL)

In propositional logic-based planning we have seen the combinatorial explosion problem that results from the need to include in the reasoning / inference step all possible states and actions combinations over time. To avoid such explosion, with obvious benefits to the planning efficiency, we introduce a language called Planning Domain Definition Language (PDDL) that allows for compressive expressiveness at the action space via *action schemas* as we will see shortly. PDDL is positioned at the input of the *domain independent* planner as shown in the figure below.



Planning System: A domain-independent solver or planner takes two inputs: 1) the domain model written in a planning language and 2) a problem definition that describes the initial state i and the desired goal state g using the domain model's terminology; and produces as output a plan, that is, a sequence of actions that takes the agent from the initial state to the goal state.

PDDL expresses the four things we need to plan a sequence of actions. The set of all predicates and action schemas are defined in the **domain** file (domain.pddl) as shown next.

Domain Syntax	Description
Types	A description of the possible *types* of objects in the world. A type can inherit from another type.
Constants	The set of *constants*, which are objects which appear in all problem instances of this domain.

Domain Syntax	Description
Predicates	A predicate is the part of a sentence or clause containing a verb and stating something about the subject. Each predicate is described by a name and a signature, consisting of an ordered list of types. Properties of the objects (contained in the problem specification) we are interested in. Each property evaluates to TRUE or FALSE. The domain also describes a set of derived predicates, which are predicates associated with a logical expression. The value of each derived predicate is computed automatically by evaluating the logical expression associated with it.
Actions / Operators	Actions are described by action schemas that effectively define the functions needed to do problem-solving search. The schema consist of the *name*, the signature or the list of all the boolean *variables* that are *ground* and functionless, a *precondition* and *effect*. The signature is now an ordered list of named parameters, each with a type.
	The **precondition** is a logical formula, whose basic building blocks are the above mentioned predicates, combined using the standard first order logic logical connectives. The predicates can only be parametrized by the operator parameters, the domain constraints, or, if they appear within the scope of a forall or exists statement, by the variable introduced by the quantifier.
	The **effect** is similar, except that it described a partial assignment, rather than a formula, and thus can not contain any disjunctions.

For the famous Blocks world shown below where a robotic arm must reason to stack blocks according what the goal is, we list the corresponding domain PDDL specification.

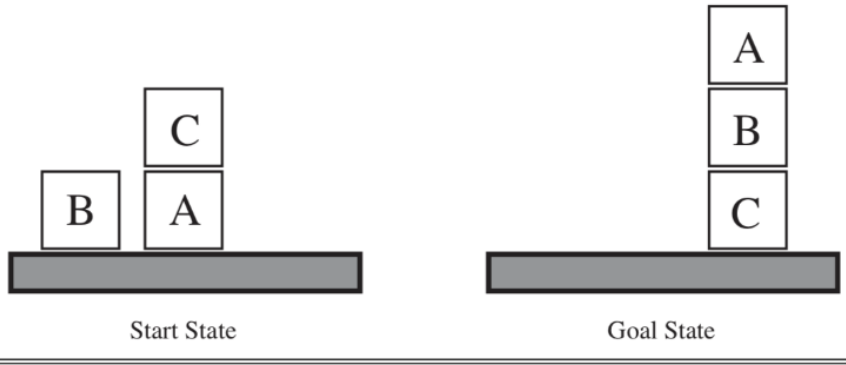


Figure 10.4 Diagram of the blocks-world problem in Figure 10.3.

In natural language, the rules are:

- Blocks are picked up and put down by the arm
- Blocks can be picked up only if they are clear, i.e., without any block on top
- The arm can pick up a block only if the arm is empty, i.e., if it is not holding another block, i.e., the arm can be pick up only one block at a time

- The arm can put down blocks on blocks or on the table

Problem Syntax	Description
Initial State	Each state is represented as conjunction of ground boolean variables. For example, $On(Box1,Table2)∧On(Box2,Table2)isastateexpression.\mathtt{Box_1}isdistinctthan\mathtt{Box_2}$. All fluents that are not specified are assumed to be FALSE.
Goal	All things we want to be TRUE. The goal is like a precondition - a conjunction of literals that may contain variables.