Planning for a Kitting Workstation

Zeid Kootbally
National Institute of Standards and Technology
zeid.kootbally@nist.gov

October 19, 2012

Contents

1 The Kitting Domain

The foundation for the knowledge representation is domain specific information that is produced by an expert in the particular field of study. This includes information on items ranging from what actions and attributes are relevant, to what the necessary conditions are for an action to occur and what the likely results of the action are. We have chosen to encode this basic information in a formalism know as a state variable representation (SVR) [1]. This information will then flow up the abstraction and be transformed into the ontology, planning language, and robot language.

Before building a SVR, the domain for kitting needs to be specified. The domain for kitting contains some fixed equipment: a robot, a work table, end effectors, end effector holders, and an end effector changing station. Items that enter the workstation include kit trays, boxes in which to put kit instances, boxes that contain empty kit trays, and part supplies. Items that leave the workstation may be boxes with finished kits (kit instances) inside, empty part trays, empty boxes. An external agent is responsible of moving the items that leave the workstation. We assume that the workstation has only one work table, one changing station, and one robot.

2 State-Variable Representation

In a SVR, each state is represented by a tuple of values of n state variables $\{x_1, \ldots, x_n\}$, and each action is represented by a partial function that maps this tuple into some other tuple of values of the n state variables.

To build the SVR, the group has taken a very systematic approach of identifying and modeling the concepts. Because the industrial robot field is so broad, the group decided to limit its efforts to a single type of operation, namely kitting. A scenario was developed that described, in detail, the types of operations that would be performed in kitting, the sequencing of steps, the parts and machines that were needed, constraints on the process such as pre- and post-conditions, etc. For this scenario, a set of concepts were extracted and defined. These concepts served as the initial requirements for the kitting SVR. The concepts were then modeling in our SVR, building off of the definitions and relationships that were identified in the scenario. A SVR relies on the elements of constant variable symbols, object variable symbols, state variable symbols, rigid relations, and planning operators. These are defined for the kitting domain in the rest of this section.

2.1 Constant Variable Symbols

For the kitting domain, there is a finite set of constant variable symbols that must be represented. In the SVR, constant variable symbols are partitioned into disjoint classes corresponding to the objects of the domain. The finite set of all constant variable symbols in the kitting domain is partitioned into the following sets:

- A set of *Part* $\{part\_a\_1, part\_a\_2, \ldots\}$: A *Part* is the basic item that will be used to fill a kit.
- A set of *PartsTray* $\{part\_a\_tray, part\_b\_tray, \ldots\}$: *Parts* arrive at the workstation in *PartsTrays*. Each *Part* is at a known position in the *PartsTray*. Each *PartsTray* contains one type of *Part*.
- A set of *KitTray* $\{kit\_tray\_1, kit\_tray\_2, \ldots\}$: A *KitTray* can hold *Parts* in known positions.
- A set of *Kit* $\{kit\_1, kit\_2, \ldots\}$: A *Kit* consists of a *KitTray* and, possibly, some *Parts*. A *Kit* is empty when it does not contain any *Part* and finished when it contains all the *Parts* that constitute a kit.
- A symbol *WorkTable* $work\_table\_1$: A *WorkTable* is an area in the kitting workstation where *KitTrays* are placed to build *Kits*.
- A set of *LargeBoxWithKits* $\{finished\_kit\_receiver\_1, finished\_kit\_receiver\_2, \ldots\}$: A *LargeBoxWithKits* contains only finished *Kits*.
- A set of *LargeBoxWithEmptyKitTrays* $\{empty\_kit\_tray\_supply\_1, empty\_kit\_tray\_supply\_2, \ldots\}$: A *LargeBoxWithEmptyKitTrays* is a box that contains only empty *KitTrays*.
- A set of *Robot* $\{robot\_1, robot\_2, \ldots\}$: A *Robot* in the kitting workstation is a robotic arm that can move objects in order to build *Kits*.
- A set of *EndEffector* $\{part\_gripper, tray\_gripper, \ldots\}$: *EndEffectors* are used in a kitting workstation to manipulate *Parts, PartsTrays, KitTrays,* and *Kits*. A *EndEffector* is attached to a *Robot*.
- A set of *EndEffectorHolder* $\{part\_gripper\_holder, tray\_gripper\_holder, \ldots\}$: An *EndEffectorHolder* is a storage unit that holds one type of *EndEffector*.
- A symbol *EndEffectorChangingStation* $changing\_station\_1$: An *EndEffectorChangingStation* is made up of *EndEffectorHolders*.

2.2 Object Variable Symbols

Object variable symbols are typed variables which range over a class or the union of classes of constant variable symbols. Examples of object variable symbols are $r \in$ *Robots*, $kt \in$ *KitTrays*, etc.

2.3 State Variable Symbols

A state variable symbol is defined as follows: $\mathrm{x} : \mathrm{A}_1 \times \cdots \times \mathrm{A}_\mathrm{i} \times \mathrm{S} \rightarrow \mathrm{B}_1 \cup \cdots \cup \mathrm{B}_\mathrm{j}$ ($i, j \geq 1$) is a function from the set of states (S) and at least one set of constant variable symbols $\mathrm{A}_1 \times \cdots \times \mathrm{A}_\mathrm{i}$ into a set of constant variable symbols $\mathrm{B}_1 \cup \cdots \cup \mathrm{B}_\mathrm{j}$.

The use of state variable symbols reduces the possibility of inconsistent states and generates a smaller state space. The following state variable symbols are used in the kitting domain:

- endeffector-location
  EndEffector$\times$S $\rightarrow$*Robot* $\cup$ *EndEffectorHolder*: designates the location of an *EndEffector* in the workstation. An *EndEffector* is either attached to a *Robot* or placed in an *EndEffectorHolder*.

- robot-with-endeffector
  Robot$\times$S $\rightarrow$*EndEffector* $\cup$ {*nil*}: designates the *EndEffector* attached to a *Robot* if there is one attached, otherwise *nil*.

- on-worktable
  WorkTable$\times$S $\rightarrow$*Kit* $\cup$ *KitTray* $\cup$ {*nil*}: designates the object placed on the *WorkTable*, i.e., a *Kit*, a *KitTray*, or nothing (*nil*).

- kit-location
  Kit$\times$S $\rightarrow$*LargeBoxWithKits* $\cup$ *WorkTable* $\cup$ *Robot*: designates the different possible locations of a *Kit* in the workstation, i.e., in a *LargeBoxWithKits*, on the *WorkTable*, or being held by a *Robot*.

- kittray-location
  KitTray$\times$S $\rightarrow$*LargeBoxWithEmptyKitTrays* $\cup$ *Robot* $\cup$ *WorkTable*: designates the different possible locations of a *KitTray* in the workstation, i.e., in a *LargeBoxWithEmptyKitTrays*, on a *WorkTable* or being held by a *Robot*.

- part-location
  Part$\times$S $\rightarrow$*PartsTray* $\cup$ *Kit* $\cup$ *Robot*: designates the different possible locations of a *Part* in the workstation, i.e., in a *PartsTray*, in a *Kit*, or being held by a *Robot*.

- robot-holds
  Robot$\times$S $\rightarrow$*KitTray* $\cup$ *Kit* $\cup$ *Part* $\cup$ {*nil*}: designates the object being held by a *Robot*, i.e., a *KitTray*, a *Kit*, a *Part*, or nothing (*nil*). It is assumed that the *Robot* is already equipped with the appropriate *EndEffector*.

- lbwk-full
  LargeBoxWithKits$\times$S $\rightarrow$ {0} $\cup$ {1}: designates if a *LargeBoxWithKits* is full (1) or not (0).

- lbwekt-empty
  LargeBoxWithEmptyKitTrays$\times$S $\rightarrow$ $\{0\} \cup \{1\}$: designates if a *LargeBoxWithEmptyKitTrays* is empty (1) or not (0).

- partstray-empty
  PartsTray$\times$S $\rightarrow$ $\{0\} \cup \{1\}$: designates if a *PartsTray* is empty (1) or not (0).

- endffector-type
  EndEffector$\times$S $\rightarrow$*KitTray* $\cup$ *Kit* $\cup$ *Part*: designates the type of object an *EndEffector* can hold, i.e., a *KitTray*, a *Kit*, or a *Part*.

- endeffectorholder-holds-endeffector
  EndEffectorHolder$\times$S $\rightarrow$*EndEffector* $\cup$ $\{nil\}$: designates wether an *EndEffectorHolder* is holding an *EndEffector* or nothing (*nil*).

- endeffectorholder-location
  EndEffectorHolder$\times$S $\rightarrow$*EndEffectorChangingStation*: designates the *EndEffectorChangingStation* where the *EndEffectorHolder* is located.

- endeffectorchangingstation-has-endeffectorholder
  EndEffectorChangingStation$\times$S $\rightarrow$*EndEffectorHolder*: designates the *EndEffectorHolder* the *EndEffectorChangingStation* contains.

- found-part
  PartsTray$\times$S $\rightarrow$*Part* $\cup$ $\{nil\}$: designates wether a *Part* is found in a *PartsTray* or not (*nil*).

- origin-part
  Part$\times$S $\rightarrow$*PartsTray*: designates the *PartsTray* where the *Part* is found.

2.4 Planning Operators and Actions

The planning operators presented in this section are expressed in classical representation instead of state variable representation. In classical representation, states are represented as sets of logical atoms (predicates) that are true or false within some interpretation. Actions are represented by planning operators that change the truth values of these atoms.

2.4.1 Convert State Variable Symbols to Atoms

In order to express actions with predicates, the state variable symbols (SVS) presented in Section 2.3 are converted into predicates (PRED). A state variable symbol can be split into multiple predicates as follows:

- SVS

- ☐ PRED1 (*param1*,*param2*, …)
- ☐ PRED2 (*param1*,*param2*, …)
- ☐ …

The state variable symbols and their corresponding predicates for the kitting domain are presented below:

- ■ endeffector-location
  - ☐ eff-location-robot(*EndEffector*,*Robot*) ;TRUE iff *EndEffector* is attached to *Robot*
  - ☐ eff-location-endeffectorholder(*EndEffector*,*EndEffectorHolder*) ;TRUE iff *EndEffector* is in *EndEffectorHolder*
- ■ robot-with-endeffector
  - ☐ robot-with-endeffector(*Robot*,*EndEffector*) ;TRUE iff *Robot* is equipped with *EndEffector*
  - ☐ robot-with-no-endeffector(*Robot*) ;TRUE iff *Robot* is not equipped with any *EndEffector*
- ■ on-worktable
  - ☐ on-worktable-kit(*WorkTable*,*Kit*) ;TRUE iff *Kit* is on the *WorkTable*
  - ☐ on-worktable-kittray(*WorkTable*,*KitTray*) ;TRUE iff *KitTray* is on the *WorkTable*
  - ☐ worktable-empty(*WorkTable*) ;TRUE iff there is nothing on the *WorkTable*
- ■ kit-location
  - ☐ kit-location-lbwk(*Kit*,*LargeBoxWithKits*) ;TRUE iff *Kit* is in the *LargeBoxWithKits*
  - ☐ kit-location-worktable(*Kit*,*WorkTable*) ;TRUE iff *Kit* is on the *WorkTable*
  - ☐ kit-location-robot(*Kit*,*Robot*) ;TRUE iff *Kit* is being held by the *Robot*
- ■ kittray-location
  - ☐ kittray-location-lbwekt(*KitTray*,*LargeBoxWithEmptyKitTrays*) ;TRUE iff *KitTray* is in the *LargeBoxWithEmptyKitTrays*
  - ☐ kittray-location-robot(*KitTray*,*Robot*) ;TRUE iff *KitTray* is being held by the *Robot*
  - ☐ kittray-location-worktable(*KitTray*,*WorkTable*) ;TRUE iff *KitTray* is on the *WorkTable*
- ■ part-location
  - ☐ part-location-partstray(*Part*,*PartsTray*) ;TRUE iff *Part* is in the *PartsTray*

- ☐ part-location-kit(*Part*,*Kit*) ;TRUE iff *Part* is in the *Kit*
- ☐ part-location-robot(*Part*,*Robot*) ;TRUE iff *Part* is being held by the *Robot*

- ■ robot-holds
  - ☐ robot-holds-kittray(*Robot*,*KitTray*) ;TRUE iff *Robot* is holding a *KitTray*
  - ☐ robot-holds-kit(*Robot*,*Kit*) ;TRUE iff *Robot* is holding a *Kit*
  - ☐ robot-holds-part(*Robot*,*Part*) ;TRUE iff *Robot* is holding a *Part*
  - ☐ robot-empty(*Robot*) ;TRUE iff *Robot* is not holding anything
- ■ lbwk-full
  - ☐ lbwk-not-full(*LargeBoxWithKits*) ;TRUE iff *LargeBoxWithKits* is not full
- ■ lbwekt-empty
  - ☐ lbwekt-not-empty(*LargeBoxWithEmptyKitTrays*) ;TRUE iff *LargeBoxWithEmptyKitTrays* is not empty
- ■ partstray-empty
  - ☐ partstray-not-empty(*PartsTray*) ;TRUE iff *PartsTray* is not empty
- ■ endeffector-type
  - ☐ endeffector-type-kittray(*EndEffector*,*KitTray*) ;TRUE iff *EndEffector* is capable of holding a *KitTray*
  - ☐ endeffector-type-kit(*EndEffector*,*Kit*) ;TRUE iff *EndEffector* is capable of holding a *Kit*
  - ☐ endeffector-type-part(*EndEffector*,*Part*) ;TRUE iff *EndEffector* is capable of holding a *Part*
- ■ endeffectorholder-holds-endeffector
  - ☐ endeffectorholder-holds-endeffector(*EndEffectorHolder*,*EndEffector*) ;TRUE iff *EndEffectorHolder* is holding *EndEffector*
  - ☐ endeffectorholder-empty(*EndEffectorHolder*) ;TRUE iff *EndEffectorHolder* is empty (not holding a *EndEffector*)
- ■ endeffectorholder-location
  - ☐ endeffectorholder-location(*EndEffectorHolder*,*EndEffectorChangingStation*) ;TRUE iff *EndEffectorHolder* is in *EndEffectorChangingStation*
- ■ endeffectorchangingstation-has-endeffectorholder

- ☐ endeffectorchangingstation-has-endeffectorholder(*EndEffectorChangingStation,EndEffectorHold*
  ;TRUE iff *EndEffectorChangingStation* contains *EndEffectorHolder*

- found-part
  - ☐ found-part(*Part,PartsTray*) ;TRUE iff *Part* is found in *PartsTray*

- origin-part
  - ☐ origin-part(*Part,PartsTray*) ;TRUE iff *Part* is from *PartsTray* at the initial state of the environment.

2.4.2 Planning Operators

In classical planning, a planning operator [1] is a triple *o=(name(o), precond(o), effects(o))* whose elements are as follows:

- name(o) is a syntactic expression of the form $n(x_1, \ldots, x_k)$, where n is a symbol called an operator symbol, $x_1, \ldots, x_k$ are all of the object variable symbols that appear anywhere in *o*, and n is unique (i.e., no two operators can have the same operator symbol).
- precond(o) and effects(o) are sets of literals (i.e., atoms and negations of atoms). Literals that are true in precond(o) but false in effects(o) are removed by using negations of the appropriate atoms.

Our kitting domain is composed of ten operators which are defined below.

1. *take-kit-tray*(r,kt,$lbwekt$,eff,$wtable$): The *Robot* r equipped with the *EndEffector* eff picks up the *KitTray* kt from the *LargeBoxWithEmptyKitTrays* $lbwekt$.

preconditions	*effects*
rhold-empty(r),	¬rhold-empty(r),
lbwekt-not-empty($lbwekt$),	kit-tray-location(kt,r),
r-with-eff(r,eff),	rhold(r,kt),
kit-tray-location(kt,$lbwekt$),	¬kit-tray-location(kt,$lbwekt$)
eff-location(eff,r),	
worktable-empty($wtable$),	
efftype(eff,kt)	

   - *preconditions*
   - *effects*

2. *put-kit-tray*(r,kt,$wtable$): The *Robot* r puts down the *KitTray* kt on the *WorkTable* $wtable$.

precond	*effects*
kit-tray-location(kt,r),	¬kit-tray-location(kt,r),
rhold(r,kt),	¬rhold(r,kt),
worktable-empty($wtable$)	¬worktable-empty($wtable$),
	kit-tray-location(kt,$wtable$),
	rhold-empty(r),
	onworktable($wtable$,kt)

3. ***take-kit***(r,kit,$wtable$,eff): The ***Robot*** r equipped with the ***EndEffector*** eff picks up the ***Kit*** kit from the ***WorkTable*** $wtable$.

precond	*effects*
kit-location(kit,$wtable$),	¬kit-location(kit,$wtable$),
rhold-empty(r),	¬rhold-empty(r),
onworktable($wtable$,kit),	¬onworktable($wtable$,kit),
r-with-eff(r,eff),	kit-location(kit,r),
efftype(eff,kit)	rhold(r,kit),
	worktable-empty($wtable$)

4. ***put-kit***(r,kit,$lbwk$): The ***Robot*** r puts down the ***Kit*** kit in the ***LargeBoxWithKits*** $lbwk$.

precond	*effects*
kit-location(kit,r),	¬kit-location(kit,r),
rhold(r,kit),	¬rhold(r,kit),
lbwk-not-full($lbwk$)	kit-location(kit,$lbwk$),
	rhold-empty(r)

5. ***take-part***(r,$part$,pt,eff,$wtable$,kit): The ***Robot*** r uses the ***EndEffector*** eff to pick up the ***Part*** $part$ from the ***PartTray*** pt. The ***Kit*** kit must ***a priori*** be on the $wtable$.

precond	*effects*
part-location($part$,pt),	¬part-location($part$,pt),
eff-location(eff,r),	rhold(r,$part$),
rhold-empty(r),	¬rhold-empty(r),
r-with-eff(r,eff),	part-location($part$,r)
onworktable($wtable$,$kins$),	
kit-location(kit,$wtable$),	
efftype(eff,$part$),	
part-tray-not-empty(pt)	

6. ***put-part***(r,$part$,kit,$wtable$): The ***Robot*** r puts down the ***Part*** $part$ in the ***Kit*** kit.

precond	*effects*
part-location($part$,r),	¬part-location($part$,r),
rhold(r,$part$),	¬rhold(r,$part$),
onworktable($wtable$,$kins$),	part-location($part$,kit),
kit-location(kit,$wtable$)	rhold-empty(r)

7. *attach-eff*(r,eff,$effh$): The *Robot* r attaches the *EndEffector* eff which is situated in the *EndEffectorHolder* $effh$.

precond	*effects*
eff-location(eff,$effh$),	¬eff-location(eff,$effh$),
r-no-eff(r),	¬r-no-eff(r),
effhold-eff($effh$,eff)	¬effhold-eff($effh$,eff),
¬effh-empty($effh$)	rhold-empty(r),
	eff-location(eff,r),
	r-with-eff(r,eff),
	effh-empty($effh$)

8. *remove-eff*(r,eff,$effh$): The *Robot* r removes the *EndEffector* eff and puts it in the *EndEffectorHolder* $effh$.

precond	*effects*
eff-location(eff,r),	¬eff-location(eff,r),
r-with-eff(r,eff),	¬r-with-eff(r,eff),
rhold-empty(r)	eff-location(eff,$effh$),
	effhold-eff($effh$,eff),
	r-no-eff(r)

9. *create-kit*(kit,kt,$wtable$): The *KitTray* kt is converted to the *Kit* kit once the *KitTray* kt is on the *WorkTable* $wtable$.

precond	*effects*
onworktable($wtable kt$)	¬onworktable($wtable$,kt),
	kit-location(kit,$wtable$),
	onworktable($wtable$,kit)

2.4.3 Actions

An action *a* can be obtained by substituting the object variable symbols that appear anywhere in the operator with constant variable symbols. For instance, the operator *take-p*(r,$part$,pt,eff) in the kitting domain can be translated into the action *take-p*(r_1,$part_1$,pt_1,eff_2) where r_1, $part_1$, pt_1, and eff_2 are constant variable symbols in the classes *Robots*, *Parts*, *PartsTrays*, and *EndEffectors*, respectively.

3 Kitting Problem

The kitting problem is quite complex and contains far too many states to represent them all explicitly. Using an example of kit to build, this section will only describe the initial and goal states explicitly. The operators detailed in Section 2.4 are used to generate the other states as needed.

In this example, the *Robot* has to build a kit that contains two *Parts* of type A, one *Part* of type B and one *Part* of type C. The kitting process is completed once the kit is placed in the *LargeBoxWithKits*.

Constant Variable Symbols The kitting domain proposed for this example (Figure 1) contains:

- One *Robot*: $robot\_1$
- One *KitTray*: kit_tray_1
- One *LargeBoxWithEmptyKitTrays*: $empty\_kit\_tray\_supply$
- One *LargeBoxWithKits*: $finished\_kit\_receiver$
- One *WorkTable*: $work\_table\_1$
- Three *PartsTrays*: $part\_a\_tray$, $part\_b\_tray$, and $part\_c\_tray$
- *Parts*: $part\_a\_1$, $part\_a\_2$, $part\_b\_1$, and $part\_c\_1$
- Two *VacuumEffectorSingleCup*: $part\_gripper$ and $tray\_gripper$
- Two *EndEffectorHolders*: $part\_gripper\_holder$ and $tray\_gripper\_holder$
- Since a *Kit* is by definition a *KitTray* that contains *Parts*, the kitting domain also contains a constant variable symbol kit_1 from *Kit*

3.1 State Variable Symbols

The state variable symbols for the kitting domains are the ones defined in section 2.3.

3.2 Rigid Relations

As stated in section **??**, the kitting domain has two rigid relations: efftype and effhold-eff that can be stated as follows:

- efftype($part\_gripper$,$part\_a\_1$)

- efftype($part\_gripper$,$part\_a\_2$)
- efftype($part\_gripper$,$part\_b\_1$)
- efftype($part\_gripper$,$part\_c\_1$)
- efftype($tray\_gripper$,kit_tray_1)
- efftype($tray\_gripper$,kit_1)

In the same way, effhold-eff can be stated as follows:

- effhold-eff($part\_gripper\_holder$,$part\_gripper$)
- effhold-eff($tray\_gripper\_holder$,$tray\_gripper$)

3.3 Initial State

The initial state s_0 (Figure 1) defines the predicates that are true in the kitting workstation. s_0 is represented in Table 1.

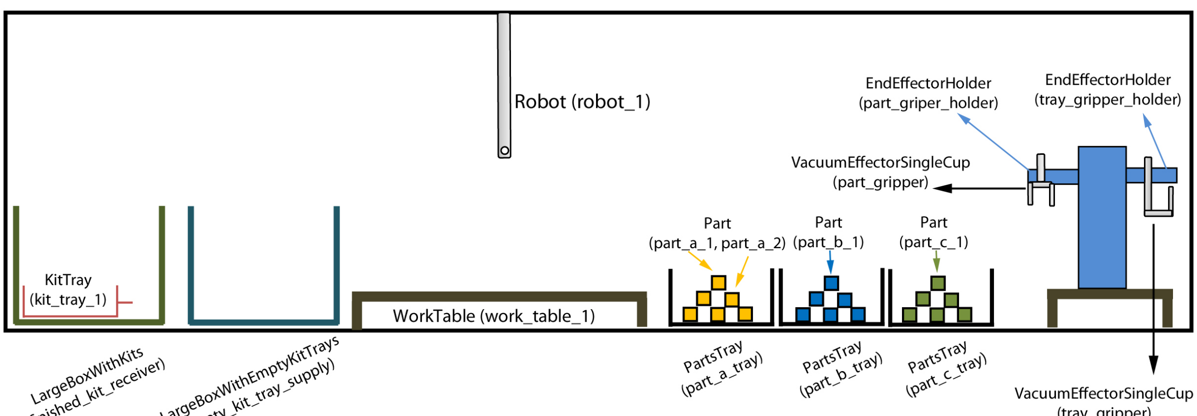

Figure 1: Initial state.

3.4 Goal State

The goal state s_G (Figure 2) defines the predicates that are true in the final state. In the goal state, *Parts* $part\_a\_1$, $part\_a\_2$, $part\_b\_1$, and $part\_c\_1$ are in the *Kit* kit_1. kit_1 is placed in the *LargeBoxWithKits* $finished\_kit\_receiver$. s_G is represented in Table 2.

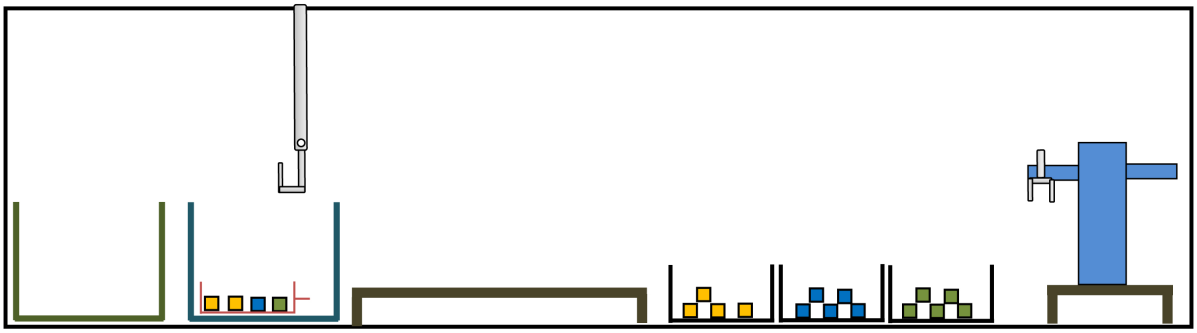

Figure 2: Goal state.

4 Planning Language

The Planning Domain Definition Language (PDDL) [2] is an attempt by the domain independent planning community to formulate a standard language for planning. A community of planning researchers has been producing planning systems that comply with this formalism since the first International Planning Competition held in 1998. This competition series continues today, with the seventh competition being held in 2011. PDDL is constantly adding extensions to the base language in order to represent more expressive problem domains. Our work is based on PDDL Version 3.

By placing our knowledge in a PDDL representation, we enable the use of an entire family of open source planning systems. Each PDDL file-set consists of two files that specify the domain and the problem.

Table 1: Initial State s_0

r-no-eff($robot\_1$)	kit-tray-location($robot\_1$,$empty\_kit\_tray\_supply$)
lbwekt-not-empty($empty\_kit\_tray\_supply$)	part-location($part\_a\_1$,$part\_a\_tray$)
lbwek-not-full($finished\_kit\_receiver$)	part-location($part\_a\_2$,$part\_a\_tray$)
part-tray-not-empty($part\_a\_tray$)	part-location($part\_b\_1$,$part\_b\_tray$)
part-tray-not-empty($part\_b\_tray$)	part-location($part\_c\_1$,$part\_c\_tray$)
part-tray-not-empty($part\_c\_tray$)	efftype($part\_gripper$,$part\_a\_1$)
eff-location($part\_gripper$,$part\_gripper\_holder$)	efftype($part\_gripper$,$part\_a\_2$)
eff-location($tray\_gripper$,$tray\_gripper\_holder$)	efftype($part\_gripper$,$part\_b\_1$)
effhhold-eff($part\_gripper\_holder$,$part\_gripper$)	efftype($part\_gripper$,$part\_c\_1$)
effhhold-eff($tray\_gripper\_holder$,$tray\_gripper$)	efftype($tray\_gripper$,kit_tray_1)
worktable-empty($work\_table\_1$)	efftype($tray\_gripper$,kit_1)

Table 2: Final State s_G

part-location($part\_a\_1$,kit_1)
part-location($part\_a\_2$,kit_1)
part-location($part\_b\_1$,kit_1)
part-location($part\_c\_1$,kit_1)
kit-location(kit_1,$finished\_kit\_receiver$)

4.1 The PDDL Domain File

The PDDL domain file is composed of four sections that include requirements, types and constants, predicates, and actions. This file may be automatically generated from a combination of information that is contained in the OWL-S process specification file and the OWL Kitting Ontology file.

The requirements section specifies which extensions this problem domain relies on. The planning system can examine this statement to determine if it is capable of solving problems in this domain. In PDDL, all variables that are used in the domain must be typed. Types are defined in the types section. It is also possible to have constants that specify that all problems will share this single value. For example, in the simplest kitting workstation we will have a single ***Robot*** $robot\_1$. Predicates specify relationships between instances. For example, an instance of a *KitTray*, kit_tray_1, can have a physical location and contains instances of *Parts*. The final section of the PDDL domain file is concerned with actions. An action statement specifies a way that a planner affects the state of the world. The statement includes parameters, preconditions, and effects. The preconditions dictate items that must be initially true for the action to be legal. The effect equation dictates the changes in the world that will occur due to the execution of the action. The components of the PDDL domain file have their source in the State Variable Representation. The PDDL domain file for kitting can be found in Appendix A.

4.2 PDDL Problem File

The second file of the PDDL file-set is a problem file. The problem file specifies information about the specific instance of the given problem. This file contains the initial conditions and definition of the world (in the init section) and the final state that the world must be brought to (in the goal section). The PDDL problem file for kitting can be found in Appendix B.

5 Planner

This section describes the steps to install and run a planner on the PDDL domain and problem files in order to generate a plan. The planner used is Randward (`http://www.plg.inf.uc3m.es/ipc2011-deterministic/ParticipatingPlanners`), a sequential satisficing planner.

5.1 Install the Planner

The planner runs on a Linux machine and can be retieved via Subversion:

- svn co svn://svn@pleiades.plg.inf.uc3m.es/ipc2011/data/planners/seq-sat/seq-sat-randward/

5.2 Compile the Planner

To compile the planner, one should use:

- ./build

5.3 Run the Planner

To run the planner, the path to the PDDL domain and problem files should be identified. The format of the PDDL files must be `.pddl`. The PDDL domain and problem files are described in Appendix A and Appendix B, respectively. The following command run the planner on the PDDL files. Note that result.txt is the output file containing the plan. If no plan is found, result.txt will not be generated.

- ./plan kitting-domain.pddl kitting-problem.pddl result.txt

The output file generated for the kitting domain and problem files can be found in Appendix C

6 The Generator

The Generator tool is a graphical user interface developed in Java, allowing the user to store data from OWL files into a MySQL database. This tool also permits the user to query the database using the C++ function calls. The tool Generator is composed of the following functionalities:

1. Convert OWL documents into SQL syntaxes (OWL to SQL).
2. Translate SQL syntaxes to OWL language in order to modify an OWL document (SQL to OWL).

3. Convert the OWL language into C++ classes (OWL to C++).

To date, only steps 1. and 3. have been implemented and will be covered in this document.

6.1 Prequisites

The description of the Generator tool is given for a Ubuntu Linux system. To run and use the Generator tool, different applications must be installed on the system.

6.1.1 Java Runtime Environment

The Generator tool comes as a jar file. As such, the Java Runtime Environment should be installed on your system. This application can be found at `www.oracle.com`.

6.1.2 MySQL Server and Client

The MySQL server and client should be installed and running on your system.

- *sudo apt-get update* (Update the package management tools)
- *sudo apt-get dist-upgrade* (Install the latest software)
- *sudo apt-get install mysql-server mysql-client* (Install the MySQL server and client packages). You will be asked to enter a password.

When done, you have a MySQL database ready to run. The following command will allow you to run MySQL.

- *mysql -u root -p*
- Enter the same password you used when you installed MySQL.

Finally, we need the plugin `libmysqlcppconn-dev` which allows C++ to connect to MySQL databases. It can be installed as follows:

- *sudo apt-get install libmysqlcppconn-dev*

6.2 How to Run the Generator Tool

The Generator tool can be launched using either one of these two following methods:

1. *java -jar Generator.jar*
2. Right-click on Generator.jar and select the option “Open With OpenJDK Java 6 Runtime". Note that this message will be different for future releases of the Java Runtime Environment.

6.3 Functionalities

As mentioned in the Introduction, we are covering only steps 1. and 3. in the rest of this document, i.e., *OWL to SQL* and *OWL to C++*, respectively.

6.3.1 OWL to SQL

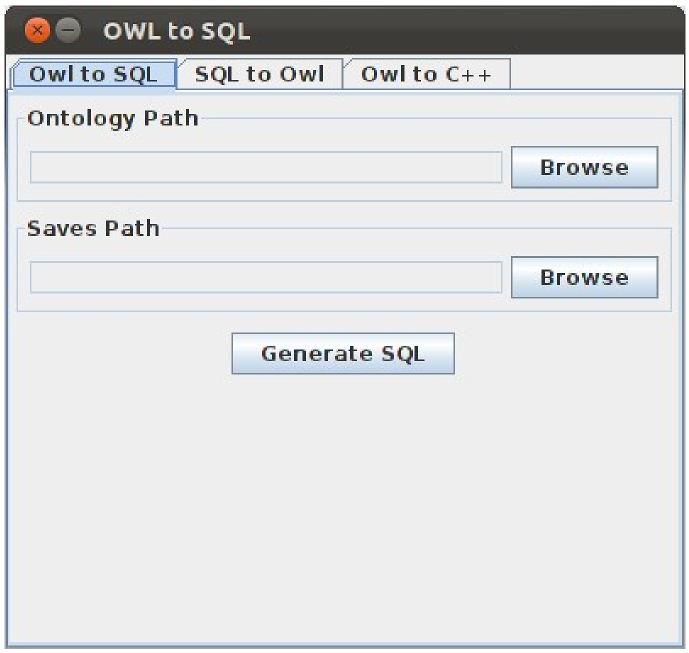

Figure 3: Owl to SQL tab.

To convert OWL classes and instances to SQL, the `Owl to SQL` tab should be selected (see Figure 3). The different fields are:

Generate SQL Files

- Ontology Path: This field requires the file `kittingInstances.owl`. Before doing so, you need to modify one line in this file. Open it with a text editor and find the line `Import(<file:kittingClasses.owl>)`. Modify this line by giving the absolute path to the file `kittingClasses.owl`. You should should have something that looks like `Import(<file:/home/username/NIST/ipmas/Generator/kittingClasses.owl>)`. When this is done, save the file, and browse to `kittingInstances.owl` using the "Browse" button.
- Browse to the directory where you want to save the SQL files.

Once the two previous steps are done, click on "Generate SQL". You should receive a message confirming the generation of the SQL files: `kittingInstances.owlCreateTable.sql` and `kittingInstances.owlInsertInto.sql`. The former is used to create tables, the latter is used to populate these tables;

SQL Tables and Insertions The next step is to create a database and to populate it.

- Connect to mysql using *mysql -u root -p*, then enter your password. You should be in the mysql shell if this succeeded (*mysql*>).
- Delete a previous database (if you already used this tool and you want to replace the existing database with this new one) : `mysql>` *DROP DATABASE OWL;* (*OWL* is the name of the old database).
- Create a database:
  - `mysql>` *CREATE DATABASE OWL;*. Here, *OWL* is the name of the database (you can use a name of your choice).
  - Before performing the following commands, we need to tell MySQL which database we are planning to work with (*OWL* in our case). This is done using:

    `mysql>` *USE OWL*
- Populate the database with tables using `kittingInstances.owlCreateTable.sql`.
  - `mysql>` *source <path>/kittingInstances.owlCreateTable.sql;*
- Populate the tables with data using `kittingInstances.owlInsertInto.sql`:
  - `mysql>` *source <path>/kittingInstances.owlInsertInto.sql;*

<path> designs the absolute path to the appropriate file.

6.3.2 OWL to C++

The “Owl to C++" tab (see Figure 4) is used to generate C++ classes and scripts allowing the connection between C++ and MySQL. The different fields are explained below:

- **Ontology Path**: This is the path to the ontology (`kittingClasses.owl` in our example).
- **Saves Path**: Directory where the C++ files and scripts will be generated.
- **Url**: This is the url of the database. It’s usually the IP address of the machine hosting the database (127.0.0.1 if it is local).
- **User name**: User name used to connect to the MySQL database.

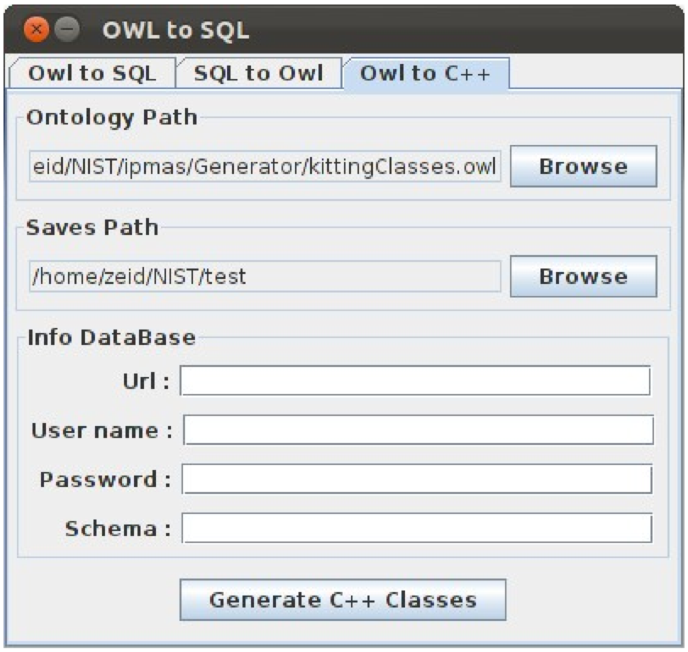

Figure 4: Owl to C++ tab.

- **Password**: Password associated to the user name to connect to the MySQL database.
- **Schema**: This is the name of the database (*OWL* in our example).

When all the fields are completed, click the “Generate C++ Classes" button to generate C++ and script files.

Appendices

A The PDDL Domain File

```
(define (domain kitting-domain)
        (:requirements :strips :typing)
        (:types
                VacuumEffectorSingleCup
                EndEffectorHolder
                Kit
                KitTray
                LargeBoxWithEmptyKitTrays
                LargeBoxWithKits
                Part
                PartsTray
                Robot
                WorkTable)

        (:predicates

        ;TRUE iff eff is attached to r
        (eff-location ?eff - VacuumEffectorSingleCup ?r - Robot)

        ;TRUE iff eff is in effh
        (eff-location ?eff - VacuumEffectorSingleCup ?effholder - EndEffectorHolder)

        ;TRUE iff r is equipped with eff
        (r-with-eff ?r - Robot ?eff - VacuumEffectorSingleCup)

        ;TRUE iff r is not equipped with any end effector
        (r-no-eff ?r - Robot)

        ;TRUE iff kins is on wtable
        (onworktable ?wtable - WorkTable ?kit - Kit)

        ;TRUE iff kt is on wtable
        (onworktable ?wtable - WorkTable ?kt - KitTray)

        ;TRUE iff there is nothing on wtable
        (worktable-empty ?wtable - WorkTable)

        ;TRUE iff kins is in lbwk
        (kit-location ?kit - Kit ?lbwk - LargeBoxWithKits)
```

```
;TRUE iff kins is on wtable
(kit-location ?kit - Kit ?wtable - WorkTable)

;TRUE iff kins is being held by r
(kit-location ?kit - Kit ?r - Robot)

;TRUE iff kt is in lbwekt
(kit-tray-location ?kt - KitTray ?lbwekt - LargeBoxWithEmptyKitTrays)

;TRUE iff kt is being held by r
(kit-tray-location ?kt - KitTray ?r - Robot)

;TRUE iff kt is on wtable
(kit-tray-location ?kt - KitTray ?wtable - WorkTable)

;TRUE iff p is in pt
(part-location ?p - Part ?pt - PartsTray)

;TRUE iff p is in kins
(part-location ?p - Part ?kit - Kit)

;TRUE iff p is being held by r
(part-location ?p - Part ?r - Robot)

;TRUE iff r is holding kt
(rhold ?r - Robot ?kt - KitTray)

;TRUE iff r is holding kins
(rhold ?r - Robot ?kit - Kit)

;TRUE iff r is holding p
(rhold ?r - Robot ?p - Part)

;TRUE iff r is not holding anything
(rhold-empty ?r - Robot)

;TRUE iff lbwk is not full
(lbwk-not-full ?lbwk - LargeBoxWithKits)

;TRUE iff lbwekt is not empty
(lbwekt-not-empty ?lbwekt - LargeBoxWithEmptyKitTrays)

;TRUE iff ?pt is not empty
(part-tray-not-empty ?pt - PartsTray)
```

```
    ;TRUE iff eff is capable of holding ?kt
    (efftype ?eff - VacuumEffectorSingleCup ?kt - KitTray)

    ;TRUE iff eff is capable of holding ?kit
    (efftype ?eff - VacuumEffectorSingleCup ?kit - Kit)

    ;TRUE iff eff is capable of holding ?p
    (efftype ?eff - VacuumEffectorSingleCup ?p - Part)

    ;TRUE iff ?effholder is holding eff
    (effhhold-eff ?effholder - EndEffectorHolder ?eff - VacuumEffectorSingleCup)

    ;TRUE iff ?effholder is empty (?effholder holds no end-effector)
    (effh-empty ?effholder - EndEffectorHolder)
)

    (:action take-kit-tray
            :parameters(
                    ?r - Robot
                    ?kt - KitTray
                    ?lbwekt - LargeBoxWithEmptyKitTrays
                    ?eff - VacuumEffectorSingleCup
                    ?wtable - WorkTable)
            :precondition(and
                    (rhold-empty ?r)
                    (lbwekt-not-empty ?lbwekt)
                    (r-with-eff ?r ?eff)
                    (kit-tray-location ?kt ?lbwekt)
                    (eff-location ?eff ?r)
                    (worktable-empty ?wtable)
                    (efftype ?eff ?kt))
            :effect(and
                    (rhold ?r ?kt)
                    (kit-tray-location ?kt ?r)
                    (not (rhold-empty ?r))
                    (not (kit-tray-location ?kt ?lbwekt))))

    (:action put-kit-tray
            :parameters(
                    ?r - Robot
                    ?kt - KitTray
                    ?wtable - WorkTable)
            :precondition
                    (and
                    (kit-tray-location ?kt ?r)
                    (rhold ?r ?kt)
                    (worktable-empty ?wtable))
```

```
        :effect(and
                (not(kit-tray-location ?kt ?r))
                (not(rhold ?r ?kt))
                (not(worktable-empty ?wtable))
                (kit-tray-location ?kt ?wtable)
                (rhold-empty ?r)
                (onworktable ?wtable ?kt)))

    (:action take-kit
        :parameters(
                ?r - Robot
                ?kit - Kit
                ?wtable - WorkTable
                ?eff - VacuumEffectorSingleCup)
        :precondition
                (and
                (kit-location ?kit ?wtable)
                (rhold-empty ?r)
                (onworktable ?wtable ?kit)
                (r-with-eff ?r ?eff)
                (efftype ?eff ?kit))
        :effect
                (and
                (kit-location ?kit ?r)
                (rhold ?r ?kit)
                (worktable-empty ?wtable)
                (not (kit-location ?kit ?wtable))
                (not (rhold-empty ?r))
                (not(onworktable ?wtable ?kit))))

(:action put-kit
    :parameters(
                ?r - Robot
                ?kit - Kit
                ?lbwk - LargeBoxWithKits)
        :precondition
                (and
                (kit-location ?kit ?r)
                (rhold ?r ?kit)
                (lbwk-not-full ?lbwk))
```

```
        :effect
                (and
                (kit-location ?kit ?lbwk)
                (rhold-empty ?r)
                (not (kit-location ?kit ?r))
                (not (rhold ?r ?kit))))

(:action take-part
        :parameters(
                ?r - Robot
                ?part - Part
                ?pt - PartsTray
                ?eff - VacuumEffectorSingleCup
                ?wtable - WorkTable
                ?kit - Kit)
        :precondition
                (and
                (part-location ?part ?pt)
                (eff-location ?eff ?r)
                (rhold-empty ?r)
                (r-with-eff ?r ?eff)
                (onworktable ?wtable ?kit)
                (kit-location ?kit ?wtable)
                (efftype ?eff ?part)
                (part-tray-not-empty ?pt))
        :effect
                (and
                (part-location ?part ?r)
                (rhold ?r ?part)
                (not (part-location ?part ?pt))
                (not (rhold-empty ?r))))

(:action put-part
        :parameters(
                ?r - Robot
                ?p - Part
                ?kit - Kit
                ?wtable - WorkTable)
        :precondition
                (and
                (part-location ?p ?r)
                (rhold ?r ?p)
                (onworktable ?wtable ?kit)
                (kit-location ?kit ?wtable))
```

```
        :effect
                (and
                (not (part-location ?p ?r))
                (not (rhold ?r ?p))
                (part-location ?p ?kit)
                (rhold-empty ?r)))

(:action attach-eff
        :parameters(
                ?r - Robot
                ?eff - VacuumEffectorSingleCup
                ?effholder - EndEffectorHolder)
        :precondition
                (and
                (eff-location ?eff ?effholder)
                (r-no-eff ?r)
                (effhhold-eff  ?effholder ?eff)
                (not (effh-empty ?effholder)))
        :effect
                (and
                (rhold-empty ?r)
                (eff-location ?eff ?r)
                (r-with-eff ?r ?eff)
                (not (eff-location ?eff ?effholder))
                (not (effhhold-eff ?effholder ?eff))
                (not (r-no-eff ?r))
                (effh-empty ?effholder)))

(:action remove-eff
        :parameters(
                ?r - Robot
                ?eff - VacuumEffectorSingleCup
                ?effholder - EndEffectorHolder)
        :precondition
                (and
                (eff-location ?eff ?r)
                (r-with-eff ?r ?eff)
                (rhold-empty ?r)
                (effh-empty ?effholder))
        :effect
                (and
                (not(eff-location ?eff ?r))
                (not(r-with-eff ?r ?eff))
                (eff-location ?eff ?effholder)
                (effhhold-eff  ?effholder ?eff)
                (r-no-eff ?r)
                (not(effh-empty ?effholder))))
```

```
        (:action create-kit
                :parameters(
                        ?kit - Kit
                        ?kt - KitTray
                        ?wtable - WorkTable)
                :precondition
                        (onworktable ?wtable ?kt)
                :effect
                        (and
                        (not(onworktable ?wtable ?kt))
                        (onworktable ?wtable ?kit)
                        (kit-location ?kit ?wtable)))
)
```

B The PDDL Problem File

```
(define (problem kitting-problem)
        (:domain kitting-domain)
        (:objects
                robot_1 - Robot
                kit_tray_1 - KitTray
                kit_1 - Kit
                empty_kit_tray_supply - LargeBoxWithEmptyKitTrays
                finished_kit_receiver - LargeBoxWithKits
                work_table_1 - WorkTable
                part_a_tray part_b_tray part_c_tray - PartsTray
                part_a_1 part_a_2 part_b_1 part_c_1 - Part
                part_gripper tray_gripper - VacuumEffectorSingleCup
                part_gripper_holder tray_gripper_holder - EndEffectorHolder
        )

        (:init
                (r-no-eff robot_1)
                (lbwekt-not-empty empty_kit_tray_supply)
                (lbwk-not-full finished_kit_receiver)
                (part-tray-not-empty part_a_tray)
                (part-tray-not-empty part_b_tray)
                (part-tray-not-empty part_c_tray)
                (eff-location part_gripper part_gripper_holder)
                (eff-location tray_gripper tray_gripper_holder)
                (effhhold-eff part_gripper_holder part_gripper)
                (effhhold-eff tray_gripper_holder tray_gripper)
                (worktable-empty work_table_1)
                (part-location part_a_1 part_a_tray)
                (part-location part_a_2 part_a_tray)
                (part-location part_b_1 part_b_tray)
                (part-location part_c_1 part_c_tray)
                (kit-tray-location kit_tray_1 empty_kit_tray_supply)
                (efftype part_gripper part_a_1)
                (efftype part_gripper part_a_2)
                (efftype part_gripper part_b_1)
                (efftype part_gripper part_c_1)
                (efftype tray_gripper kit_tray_1)
                (efftype tray_gripper kit_1))
```

```
        (:goal
          (and
            (part-location part_a_1 kit_1)
            (part-location part_a_2 kit_1)
            (part-location part_b_1 kit_1)
            (part-location part_c_1 kit_1)
            (kit-location kit_1 finished_kit_receiver)))
)
```

C The Plan

```
(attach-eff robot_1 tray_gripper tray_gripper_holder)
(take-kit-tray robot_1 kit_tray_1 empty_kit_tray_supply tray_gripper work_table_1)
(put-kit-tray robot_1 kit_tray_1 work_table_1)
(create-kit kit_1 kit_tray_1 work_table_1)
(remove-eff robot_1 tray_gripper tray_gripper_holder)
(attach-eff robot_1 part_gripper part_gripper_holder)
(take-part robot_1 part_a_1 part_a_tray part_gripper work_table_1 kit_1)
(put-part robot_1 part_a_1 kit_1 work_table_1)
(take-part robot_1 part_a_2 part_a_tray part_gripper work_table_1 kit_1)
(put-part robot_1 part_a_2 kit_1 work_table_1)
(take-part robot_1 part_b_1 part_b_tray part_gripper work_table_1 kit_1)
(put-part robot_1 part_b_1 kit_1 work_table_1)
(take-part robot_1 part_c_1 part_c_tray part_gripper work_table_1 kit_1)
(put-part robot_1 part_c_1 kit_1 work_table_1)
(remove-eff robot_1 part_gripper part_gripper_holder)
(attach-eff robot_1 tray_gripper tray_gripper_holder)
(take-kit robot_1 kit_1 work_table_1 tray_gripper)
(put-kit robot_1 kit_1 finished_kit_receiver)
```

References

[1] Nau, D., Ghallab, M., and Traverso, P., 2004. *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[2] Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D., 1998. PDDL–The Planning Domain Definition Language. Tech. Rep. CVC TR98-003/DCS TR-1165, Yale.