Planning for a Kitting Workstation

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

June 27, 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 *VacuumEffectorSingleCup* $\{part\_gripper, tray\_gripper, \ldots\}$: *VacuumEffectorSingleCups* are used in a kitting workstation to manipulate *Parts*, *PartsTrays*, *KitTrays*, and *Kits*. A *VacuumEffectorSingleCup* 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:

- efflocation: *VacuumEffectorSingleCup*$\times \mathrm{S} \rightarrow$*Robot* $\cup$ *EndEffectorHolder*: designates the location of a *VacuumEffectorSingleCup* in the workstation. A *VacuumEffectorSingleCup* is either attached to a *Robot* or placed in an *EndEffectorHolder*.
- r-eff: *Robots*$\times \mathrm{S} \rightarrow$*VacuumEffectorSingleCup* $\cup$ {*nil*}: designates the *VacuumEffectorSingleCup* attached to a *Robot* if there is one attached, otherwise *nil*.
- on-worktable: *WorkTable*$\times \mathrm{S} \rightarrow$*Kit* $\cup$ *KitTray* $\cup$ {*nil*}: designates the object placed on the *WorkTable*, i.e., a *Kit*, a *KitTray*, or nothing (*nil*).
- kitlocation: *Kit*$\times \mathrm{S} \rightarrow$*LargeBoxWithKits* $\cup$ *WorkTable* $\cup$ *Robots*: 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*.
- kittraylocation: *KitTray*$\times \mathrm{S} \rightarrow$*LargeBoxWithEmptyKitTrays* $\cup$ *Robots* $\cup$ *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*.
- partlocation: *Part*$\times \mathrm{S} \rightarrow$*PartsTray* $\cup$ *Kit* $\cup$ *Robots*: designates the different possible locations of a *Part* in the workstation, i.e., in a *PartTray*, in a *Kit*, or being held by a *Robot*.
- rhold: *Robot*$\times \mathrm{S} \rightarrow$*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 *VacuumEffectorSingleCup*.
- islbwkfull: *LargeBoxWithKits*$\times \mathrm{S} \rightarrow \{0\} \cup \{1\}$: designates if a *LargeBoxWithKits* is full (1) or not (0).
- islbwektempty: *LargeBoxWithEmptyKitTrays*$\times \mathrm{S} \rightarrow \{0\} \cup \{1\}$: designates if a *LargeBoxWithEmptyKitTrays* is empty (1) or not (0).
- isptempty: *PartsTray*$\times \mathrm{S} \rightarrow \{0\} \cup \{1\}$: designates if a *PartsTray* is empty (1) or not (0).
- efftype: *VacuumEffectorSingleCup*$\times \mathrm{S} \rightarrow$*KitTray* $\cup$ *Kit* $\cup$ *Part*: designates the type of object an *VacuumEffectorSingleCup* can hold, i.e., a *KitTray*, a *Kit*, or a *Part*.

- effhold-eff: *EndEffectorHolder*$\times$S $\rightarrow$*VacuumEffectorSingleCup* $\cup$ {*nil*}: designates the object an *EndEffectorHolder* is holding, i.e., a *VacuumEffectorSingleCup* or nothing.

2.4 Rigid Relations

efftype and effhold-eff are rigid relations since their values do not vary from one state to another. In each state, a given *EndEffector* will always hold the same type of object and a given *EndEffectorHolder* will always hold the same *EndEffectors*.

2.5 Planning Operators and Actions

The planning operators presented in this section will be expressed in classical representation instead of state variable representation. In classical representation, states are represented as sets of logical atoms that are true or false within some interpretation. Actions are represented by planning operators that change the truth values of these atoms. Predicates and actions in the domain and problem PDDL files (see Section 4) need to be expressed with logical atoms, hence the use of classical representation.

2.5.1 Convert State Variable Symbols to Atoms

In order to use sets of logical atoms, the state variable symbols (SVSs) 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 kitting are presented below:

- efflocation
  - eff-location(*VacuumEffectorSingleCup*,*Robot*) ;TRUE iff *VacuumEffectorSingleCup* is attached to *Robot*
  - eff-location(*VacuumEffectorSingleCup*,*EndEffectorHolder*) ;TRUE iff *VacuumEffectorSingleCup* is in *EndEffectorHolder*
- r-eff

- r-with-eff(*Robot,VacuumEffectorSingleCup*) ;TRUE iff *Robot* is equipped with *VacuumEffectorSingleCup*
- r-no-eff(*Robot*) ;TRUE iff *Robot* is not equipped with any *VacuumEffectorSingleCup*

■ on-worktable

- onworktable(*WorkTable,Kit*) ;TRUE iff *Kit* is on the *WorkTable*
- onworktable(*WorkTable,KitTray*) ;TRUE iff *KitTray* is on the *WorkTable*
- worktable-empty(*WorkTable*) ;TRUE iff there is nothing on the *WorkTable*

■ kitlocation

- kit-location(*Kit,LargeBoxWithKits*) ;TRUE iff *Kit* is in the *LargeBoxWithKits*
- kit-location(*Kit,WorkTable*) ;TRUE iff *Kit* is on the *WorkTable*
- kit-location(*Kit,Robot*) ;TRUE iff *Kit* is being held by the *Robot*

■ ktlocation

- kit-tray-location(*KitTray,LargeBoxWithEmptyKitTrays*) ;TRUE iff *KitTray* is in the *LargeBoxWithEmptyKitTrays*
- kit-tray-location(*KitTray,Robot*) ;TRUE iff *KitTray* is being held by the *Robot*
- kit-tray-location(*KitTray,WorkTable*) ;TRUE iff *KitTray* is on the *WorkTable*

■ partlocation

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

■ rhold

- rhold(*Robot,KitTray*) ;TRUE iff *Robot* is holding a *KitTray*
- rhold(*Robot,Kit*) ;TRUE iff *Robot* is holding a *Kit*
- rhold(*Robot,Part*) ;TRUE iff *Robot* is holding a *Part*
- rhold-empty(*Robot*) ;TRUE iff *Robot* is not holding anything

■ islbwkfull

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

■ islbwektempty

- ☐ lbwekt-not-empty(*LargeBoxWithEmptyKitTrays*) ;TRUE iff *LargeBoxWithEmptyKitTrays* is not empty

- ■ isptempty
  - ☐ part-tray-not-empty(*PartsTray*) ;TRUE iff *PartsTray* is not empty

- ■ efftype
  - ☐ efftype(*VacuumEffectorSingleCup,KitTray*) ;TRUE iff *VacuumEffectorSingleCup* is capable of holding a *KitTray*
  - ☐ efftype(*VacuumEffectorSingleCup,Kit*) ;TRUE iff *VacuumEffectorSingleCup* is capable of holding a *Kit*
  - ☐ efftype(*VacuumEffectorSingleCup,Part*) ;TRUE iff *VacuumEffectorSingleCup* is capable of holding a *Part*

- ■ effhold-eff
  - ☐ effhold-eff(*EndEffectorHolder,VacuumEffectorSingleCup*) ;TRUE iff *EndEffectorHolder* is holding *VacuumEffectorSingleCup*
  - ☐ effh-empty(*EndEffectorHolder*) ;TRUE iff *EndEffectorHolder* is empty (not holding a *VacuumEffectorSingleCup*)

2.5.2 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 nine operators which are defined below.

1. *take-kit-tray*(r,kt,$lbwekt$,eff,$wtable$): The *Robot* r equipped with the *VacuumEffectorSingleCup* eff picks up the *KitTray* kt from the *LargeBoxWithEmptyKitTrays* $lbwekt$. The *WorkTable* $wtable$ must be *a priori* empty.

precond	*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)	

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 *VacuumEffectorSingle-Cup* 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 *VacuumEffectorSingle-Cup* 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 *VacuumEffectorSingleCup* 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 *VacuumEffectorSingleCup* 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($wtablekt$)	¬onworktable($wtable$,kt), kit-location(kit,$wtable$), onworktable($wtable$,kit)

2.5.3 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 2.5 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 a *Robot* r_1, a *KitTray* kt_1, a *LargeBoxWithEmptyKitTrays* $lbwekt_1$, a *LargeBoxWithKits* $lbwk_1$, a *WorkTable* $wtable$, three *PartTrays* pt_A, pt_B, and pt_C, *Parts* $part_{A-1}$, $part_{A-2}$, $part_B$, and $part_C$, two *EndEffectors* eff_1 and eff_2, and two *EndEffectorHolders* $effh_1$ and $effh_2$. Since a *KitInstance* is by definition a *KitTray* that contains *Parts*, the kitting domain also contains a constant variable symbol $kins_1$ from *KitInstance*.

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 2.4, the kitting domain has two rigid relations: efftype and effhold-eff that can be stated as follows:

- efftype(eff_1,$part_{A-1}$)

- efftype(eff_1,$part_{A-2}$)
- efftype(eff_1,$part_B$)
- efftype(eff_1,$part_C$)
- efftype(eff_2,kt_1)
- efftype(eff_2,$kins_1$)

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

- effhold-eff($effh_1$,eff_1)
- effhold-eff($effh_2$,eff_2)

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.

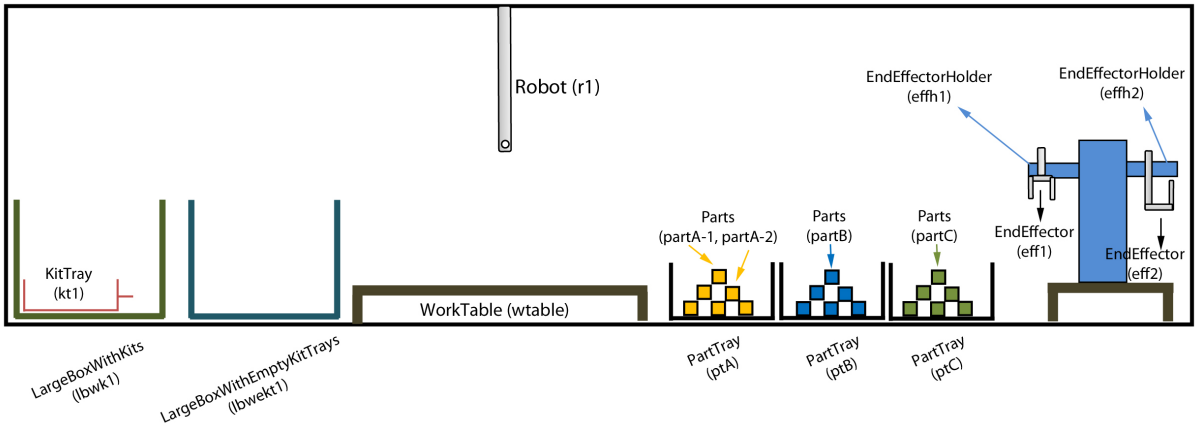

Figure 1: Initial state.

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* $part_{A-1}$, $part_{A-2}$, $part_B$, and $part_C$ are in the *KitInstance* $kins_1$. $kins_1$ is placed in the *LargeBoxWithKits* $lbwk_1$. s_G is represented in Table 2.

Table 1: Initial State s_0

r-no-eff(r_1)	ktlocation(r_1,$lbwekt_1$)
lbwekt-non-empty($lbwekt_1$)	partlocation($part_{A-1}$,pt_A)
lbwek-non-full($lbwk_1$)	partlocation($part_{A-2}$,pt_A)
part-tray-non-empty(pt_A)	partlocation($part_B$,pt_B)
part-tray-non-empty(pt_B)	partlocation($part_C$,pt_C)
part-tray-non-empty(pt_C)	efftype(eff_1,$part_{A-1}$)
efflocation(eff_1,$effh_1$)	efftype(eff_1,$part_{A-2}$)
efflocation(eff_2,$effh_2$)	efftype(eff_1,$part_B$)
effhhold-eff($effh_1$,eff_1)	efftype(eff_1,$part_C$)
effhhold-eff($effh_2$,eff_2)	efftype(eff_2,kt_1)
worktable-empty($wtable$)	efftype(eff_2,$kins_1$)

Table 2: Final State s_G

partlocation($part_{A-1}$,$kins_1$)
partlocation($part_{A-2}$,$kins_1$)
partlocation($part_B$,$kins_1$)
partlocation($part_C$,$kins_1$)
kinslocation($kins_1$,$lbwk_1$)

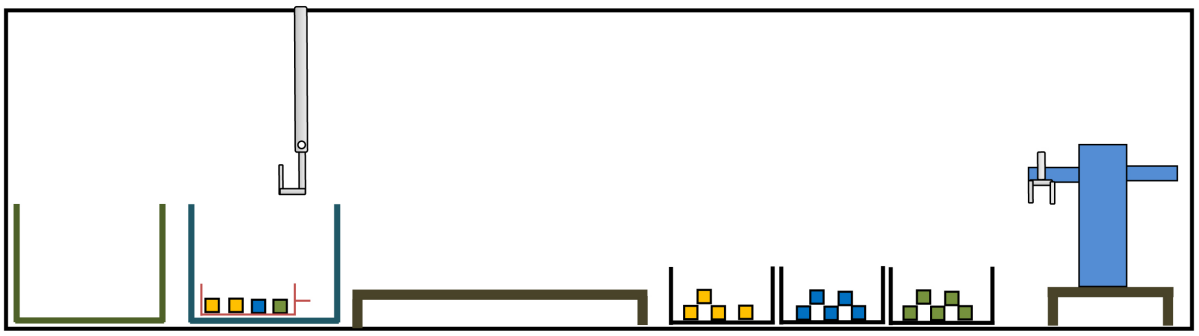

Figure 2: Goal state.

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.

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* r_1. Predicates specify relationships between instances. For example, an instance of a *KitTray*, kt_1, can have a physical location and contains instances of *Parts*, $part_A$, $part_B$, and $part_C$. 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

Appendices

A The PDDL Domain File

```
(define (domain kitting-domain)
        (:requirements :strips :typing)
        (:types
                endeffector
                endeffectorchangingstation
                endeffectorholder
                kitinstance
                kittray
                largeboxwithemptykittray
                largeboxwithkit
                part
                parttray
                robot
                worktable)

(:predicates

        ;TRUE iff eff is attached to r
        (efflocation ?eff - endeffector ?r - robot)

        ;TRUE iff eff is in effh
        (efflocation ?eff - endeffector ?effh - endeffectorholder)

        ;TRUE iff r is equipped with eff
        (r-with-eff ?r - robot ?eff - endeffector)

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

        ;TRUE iff kins is on wtable
        (onworktable ?wtable - worktable ?kins - kitinstance)

        ;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
        (kinslocation ?kins - kitinstance ?lbwk - largeboxwithkit)
```

```
;TRUE iff kins is on wtable
(kinslocation ?kins - kitinstance ?wtable - worktable)

;TRUE iff kins is being held by r
(kinslocation ?kins - kitinstance ?r - robot)

;TRUE iff kt is in lbwekt
(ktlocation ?kt - kittray ?lbwekt - largeboxwithemptykittray)

;TRUE iff kt is being held by r
(ktlocation ?kt - kittray ?r - robot)

;TRUE iff kt is on wtable
(ktlocation ?kt - kittray ?wtable - worktable)

;TRUE iff p is in pt
(partlocation ?p - part ?pt - parttray)

;TRUE iff p is in kins
(partlocation ?p - part ?kins - kitinstance)

;TRUE iff p is being held by r
(partlocation ?p - part ?r - robot)

;TRUE iff r is holding kt
(rhold ?r - robot ?kt - kittray)

;TRUE iff r is holding kins
(rhold ?r - robot ?kins - kitinstance)

;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-non-full ?lbwk - largeboxwithkit)

;TRUE iff lbwekt is not empty
(lbwekt-non-empty ?lbwekt - largeboxwithemptykittray)

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

```
    ;TRUE iff eff is for kt
    (efftype - endeffector ?kt - kittray)

    ;TRUE iff eff is holding kins
    (efftype ?eff - endeffector ?kins - kitinstance)

    ;TRUE iff eff is holding p
    (efftype ?eff - endeffector ?p - part)

    ;TRUE iff effh is holding eff
    (effhhold-eff ?effh - endeffectorholder ?eff - endeffector)
)
    (:action take-kt
            :parameters(
                    ?r - robot
                    ?kt - kittray
                    ?lbwekt - largeboxwithemptykittray
                    ?eff - endeffector
                    ?wtable - worktable)
            :precondition(and
                    (rhold-empty ?r)
                    (lbwekt-non-empty ?lbwekt)
                    (r-with-eff ?r ?eff)
                    (ktlocation ?kt ?lbwekt)
                    (efflocation ?eff ?r)
                    (worktable-empty ?wtable)
                    (efftype ?eff ?kt))
            :effect(and
                    (rhold ?r ?kt)
                    (ktlocation ?kt ?r)
                    (not (rhold-empty ?r))
                    (not (ktlocation ?kt ?lbwekt))))

    (:action put-kt
            :parameters(
                    ?r - robot
                    ?kt - kittray
                    ?wtable - worktable)
            :precondition
                    (and
                    (ktlocation ?kt ?r)
                    (rhold ?r ?kt)
                    (worktable-empty ?wtable))
```

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

(:action take-kins
        :parameters(
                ?r - robot
                ?kins - kitinstance
                ?wtable - worktable
                ?eff - endeffector)
        :precondition
                (and
                (kinslocation ?kins ?wtable)
                (rhold-empty ?r)
                (onworktable ?wtable ?kins)
                (r-with-eff ?r ?eff)
                (efftype ?eff ?kins))
        :effect
                (and
                (kinslocation ?kins ?r)
                (rhold ?r ?kins)
                (worktable-empty ?wtable)
                (not (kinslocation ?kins ?wtable))
                (not (rhold-empty ?r))
                (not(onworktable ?wtable ?kins))))

(:action put-kins
        :parameters(
                ?r - robot
                ?kins - kitinstance
                ?lbwk - largeboxwithkit)

        :precondition
                (and
                (kinslocation ?kins ?r)
                (rhold ?r ?kins)
                (lbwk-non-full ?lbwk))
```

```
        :effect
                (and
                (kinslocation ?kins ?lbwk)
                (rhold-empty ?r)
                (not (kinslocation ?kins ?r))
                (not (rhold ?r ?kins))))

(:action take-part
        :parameters(
                ?r - robot
                ?part - part
                ?pt - parttray
                ?eff - endeffector
                ?wtable - worktable
                ?kins - kitinstance)
        :precondition
                (and
                (partlocation ?part ?pt)
                (efflocation ?eff ?r)
                (rhold-empty ?r)
                (r-with-eff ?r ?eff)
                (onworktable ?wtable ?kins)
                (kinslocation ?kins ?wtable)
                (efftype ?eff ?part)
                (part-tray-non-empty ?pt))
        :effect
                (and
                (partlocation ?part ?r)
                (rhold ?r ?part)
                (not (partlocation ?part ?pt))
                (not (rhold-empty ?r))))

(:action put-part
        :parameters(
                ?r - robot
                ?p - part
                ?kins - kitinstance
                ?wtable - worktable)
        :precondition
                (and
                (partlocation ?p ?r)
                (rhold ?r ?p)
                (onworktable ?wtable ?kins)
                (kinslocation ?kins ?wtable))
```

```
        :effect
                (and
                (not (partlocation ?p ?r))
                (not (rhold ?r ?p))
                (partlocation ?p ?kins)
                (rhold-empty ?r)))

(:action attach-eff
        :parameters(
                ?r - robot
                ?eff - endeffector
                ?effh - endeffectorholder)
        :precondition
                (and
                (efflocation ?eff ?effh)
                (r-no-eff ?r)
                (effhhold-eff  ?effh ?eff)
                )
        :effect
                (and
                (rhold-empty ?r)
                (efflocation ?eff ?r)
                (r-with-eff ?r ?eff)
                (not (efflocation ?eff ?effh))
                (not (effhhold-eff  ?effh ?eff))
                (not (r-no-eff ?r))))

(:action remove-eff
        :parameters(
                ?r - robot
                ?eff - endeffector
                ?effh - endeffectorholder)
        :precondition
                (and
                (efflocation ?eff ?r)
                (r-with-eff ?r ?eff)
                (rhold-empty ?r))
        :effect
                (and
                (not(efflocation ?eff ?r))
                (not(r-with-eff ?r ?eff))
                (efflocation ?eff ?effh)
                (effhhold-eff  ?effh ?eff)
                (r-no-eff ?r)))
```

```
        (:action create-kins
                :parameters(
                        ?kins - kitinstance
                        ?kt - kittray
                        ?wtable - worktable)
                :precondition
                        (onworktable ?wtable ?kt)
                :effect
                        (and
                        (not(onworktable ?wtable ?kt))
                        (onworktable ?wtable ?kins)
                        (kinslocation ?kins ?wtable)))
)
```

B The PDDL Problem File

```
(define (problem kitting-problem)
        (:domain kitting-domain)
        (:objects
                r1 - robot
                kt1 - kittray
                kins1 - kitinstance
                lbwekt1 - largeboxwithemptykittray
                lbwk1 - largeboxwithkit
                wtable - worktable
                ptA ptB ptC - parttray
                partA1 partA2 partB partC - part
                eff1 eff2 - endeffector
                effh1 effh2 - endeffectorholder)

        (:init
                (r-no-eff r1)
                (lbwekt-non-empty lbwekt1)
                (lbwk-non-full lbwk1)
                (part-tray-non-empty ptA)
                (part-tray-non-empty ptB)
                (part-tray-non-empty ptC)
                (efflocation eff1 effh1)
                (efflocation eff2 effh2)
                (effhhold-eff effh1 eff1)
                (effhhold-eff effh2 eff2)
                (worktable-empty wtable)
                (partlocation partA1 ptA)
                (partlocation partA2 ptA)
                (partlocation partB ptB)
                (partlocation partC ptC)
                (ktlocation kt1 lbwekt1)
                (efftype eff1 partA1)
                (efftype eff1 partA2)
                (efftype eff1 partB)
                (efftype eff1 partC)
                (efftype eff2 kt1)
                (efftype eff2 kins1))
```

```
        (:goal
          (and
            (partlocation partA1 kins1)
            (partlocation partA2 kins1)
            (partlocation partB kins1)
            (partlocation partC kins1)
            (kinslocation kins1 lbwk1)))
)
```

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.