NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

Intelligent Systems Division

Knowledge Driven Planning and Modeling for Part Handling

Planning for a Kitting Workstation

Stephen Balakirsky	stephen.balakirsky@nist.gov
Zeid Kootbally	zeid.kootbally@nist.gov
Thomas Kramer	thomas.kramer@nist.gov
Anthony Pietromartire	pietromartire.anthony@nist.gov
Craig Schlenoff	craig.schlenoff@nist.gov

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) [4]. 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 kits, boxes that contain empty kit trays, and part supplies. Items that leave the workstation may be boxes with finished kits 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, 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*: A *Part* is the basic item that will be used to fill a kit.
- A set of *PartsTray*: *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*: A *KitTray* can hold *Parts* in known positions.
- A set of *Kit*: 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 set of *WorkTable*: A *WorkTable* is an area in the kitting workstation where *KitTrays* are placed to build *Kits*.
- A set of *LargeBoxWithKits*: A *LargeBoxWithKits* contains only finished *Kits*.
- A set of *LargeBoxWithEmptyKitTrays*: A *LargeBoxWithEmptyKitTrays* is a box that contains only empty *KitTrays*.
- A set of *Robot* {*robot_1*,*robot_2*,...}: A *Robot* in the kitting workstation is a robotic arm that can move objects in order to build *Kits*.
- A set of *EndEffector*: *EndEffectors* are used in a kitting workstation to manipulate *Parts*, *PartsTrays*, *KitTrays*, and *Kits*. An *EndEffector* is attached to a *Robot* in order to grasp objects.
- A set of *EndEffectorHolder*: An *EndEffectorHolder* is a storage unit that holds one type of *EndEffector*.
- A set of *EndEffectorChangingStation*: 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:
$\mathrm{x} : \mathrm{A}_1 \times \cdots \times \mathrm{A}_\mathrm{i} \times \mathrm{S} \rightarrow \mathrm{B}_1 \cup \cdots \cup \mathrm{B}_\mathrm{j} \cup \mathit{bool} \cup \{\} \cup \mathit{numeric}$ $(i, j \geq 1)$ 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 $\mathrm{B}_1 \cup \cdots \cup \mathrm{B}_\mathrm{j} \cup \mathit{bool} \cup \{\} \cup \mathit{numeric}$ where:

- $\mathrm{B}_1 \cup \cdots \cup \mathrm{B}_\mathrm{j}$ is a set of constant variable symbols
- *bool* is a boolean
- $\{\}$ is an empty set

- *numeric* is a numerical value

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×S →*Robot* ∪ *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×S →*EndEffector* ∪ {}: designates the *EndEffector* attached to a *Robot* if there is one attached, otherwise nothing.

- on-worktable
  WorkTable×S →*Kit* ∪ *KitTray* ∪ {}: designates the object placed on the *WorkTable*, i.e., a *Kit*, a *KitTray*, or nothing.

- kit-location
  Kit×S →*LargeBoxWithKits* ∪ *WorkTable* ∪ *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×S →*LargeBoxWithEmptyKitTrays* ∪ *WorkTable* ∪ *Robot*: 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×S →*PartsTray* ∪ *Kit* ∪ *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×S →*KitTray* ∪ *Kit* ∪ *Part* ∪ {}: designates the object being held by a *Robot*, i.e., a *KitTray*, a *Kit*, a *Part*, or nothing. It is assumed that the *Robot* is already equipped with the appropriate *EndEffector*.

- lbwk-full
  LargeBoxWithKits×S → *bool*: designates if a *LargeBoxWithKits* is full or not.

- lbwekt-empty
  $LargeBoxWithEmptyKitTrays \times \mathrm{S} \rightarrow$ *bool*: designates if a *LargeBoxWithEmptyKitTrays* is empty or not.

- partstray-empty
  $PartsTray \times \mathrm{S} \rightarrow$ *bool*: designates if a *PartsTray* is empty or not.

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

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

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

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

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

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

- quantity-parts-in-partstray
  $PartsTray \times \mathrm{S} \rightarrow numeric$: designates the number of parts that *PartsTray* contains.

- quantity-parts-in-kit
  $Kit \times PartsTray \times \mathrm{S} \rightarrow numeric$: designates the number of parts from *PartsTray* that *Kit* contains.

- capacity-parts-in-kit
  $Kit \times PartsTray \times \mathrm{S} \rightarrow numeric$: designates the number of parts from *PartsTray* that *Kit* **can** contain.

2.4 Predicates and Functions

In PDDL, predicates are used to encode Boolean state variables, while functions are used to model updates of numerical values [2]. This section describes the predicates and functions derived from the state variables described in section 2.3. We recall the following definition of a state variable ((section 2.3)) $\mathrm{x} : \mathrm{A}_1 \times \cdots \times \mathrm{A}_\mathrm{i} \times \mathrm{S} \rightarrow \mathrm{B}_1 \cup \cdots \cup \mathrm{B}_\mathrm{j}$ $(i, j \geq 1)$ that is used to convert state variables into predicates as follows:

- $\mathrm{A}_1 \times \cdots \times \mathrm{A}_\mathrm{i} \times \mathrm{S} \rightarrow \mathrm{B}_1 \cup \cdots \cup \mathrm{B}_\mathrm{j}$ $(i, j \geq 1)$
  - predicate_1$(\mathcal{A}, \mathcal{B})$
  - ...
  - predicate_n$(\mathcal{A}, \mathcal{B})$

Where $\mathcal{A} \in \{\mathrm{A}_1, \ldots, \mathrm{A}_\mathrm{i}\}$ and $\mathcal{B} \in \{\mathrm{B}_1, \ldots, \mathrm{B}_\mathrm{i}\}$ $(i, j \geq 1)$

2.4.1 Predicates

The state variables in our current kitting domain contains the following predicates.

- endeffector-location
  - endeffector-location-robot(*EndEffector*,*Robot*) ;TRUE iff *EndEffector* is attached to *Robot*
  - endeffector-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 an *EndEffector*)

- ■ endeffectorholder-location
  - ☐ endeffectorholder-location(*EndEffectorHolder*,*EndEffectorChangingStation*) ;TRUE iff *EndEffectorHolder* is in *EndEffectorChangingStation*

- ■ endeffectorchangingstation-has-endeffectorholder
  - ☐ endeffectorchangingstation-has-endeffectorholder(*EndEffectorChangingStation*,*EndEffect* ;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*.

2.4.2 Functions

In a planning model, numeric fluents represent function symbols that can take an infinite set of values. Introducing functions into planning not only makes it possible to deal with numerical values in a more general way than allowed for by a purely relational language but makes it possible to model operators in a more compact and sometimes also more natural way. The state variables in our current kitting domain contains the following functions.

- quantity-parts-in-partstray
  - quantity-parts-in-partstray(*PartsTray*) ;Quantity of parts in *PartsTray*
- quantity-parts-in-kit
  - quantity-parts-in-kit(*Kit*,*PartsTray*) ;Quantity of parts from *PartsTray* that is in *Kit*
- capacity-parts-in-kit
  - capacity-parts-in-kit(*Kit*,*PartsTray*) ;Quantity of parts from *PartsTray* that *Kit* **can** contain

2.5 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.

2.5.1 Planning Operators

In classical planning, a planning operator [4] is a triple o=(name(o), *preconditions*(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).

- *preconditions*(o) and *effects*(o) are sets of literals (i.e., atoms and negations of atoms). Literals that are true in *preconditions*(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-kittray(*robot*,*kittray*,*lbwekt*,*endeffector*,*worktable*): The *Robot robot* equipped with the *EndEffector endeffector* picks up the *KitTray kittray* from the *LargeBoxWithEmptyKitTrays lbwekt*.

preconditions	*effects*
robot-empty(*robot*)	¬robot-empty(*robot*)
kittray-location-lbwekt(*kittray*,*lbwekt*)	¬kittray-location-lbwekt(*kittray*,*lbwekt*)
lbwekt-not-empty(*lbwekt*)	kit-tray-location(*kittray*,*robot*)
robot-with-endeffector(*robot*,*endeffector*)	robot-holds-kittray(*robot*,*kittray*)
endeffector-location-robot(*endeffector*,*robot*)	
worktable-empty(*worktable*)	
endeffector-type-kittray(*endeffector*,*kittray*)	

- *preconditions*
  - ☐ robot-empty(*robot*): *robot* does not hold anything.
  - ☐ kittray-location-lbwekt(*kittray*,*lbwekt*): *kittray* is in *lbwekt*.
  - ☐ lbwekt-not-empty(*lbwekt*): *lbwekt* is not empty (contains at least one kit tray).
  - ☐ robot-with-endeffector(*robot*,*endeffector*): *robot* is equipped with *endeffector*.
  - ☐ endeffector-location-robot(*endeffector*,*robot*): The end effector is on the robot's arm.
  - ☐ worktable-empty(*worktable*): After picking up an empty kit tray from a large box of empty kit trays, the robot would normally place the kit tray on the work table. To put a kit tray on the work table, it is necessary that there is nothing on top of the work table. If the robot is allowed to pick up the kit tray while there is another object on the work table, the planning system may not be able to find a solution when it comes to put the kit tray on the work table. Therefore, it is necessary to check that the top of *worktable* is clear even before the robot picks up a kit tray from the large box of empty kit trays.
  - ☐ endeffector-type-kittray(*endeffector*,*kittray*): *endeffector* in the robot's arm must be capable of handling *kittray*.
- *effects*

- ☐ ¬robot-empty(*robot*): *robot*'s end effector is no longer empty since it contains *kittray*.
- ☐ ¬kittray-location-lbwekt(*kittray*,*lbwekt*): *kittray* is no longer in *lbwekt* since it is in the robot's end effector.
- ☐ kit-tray-location(*kittray*,*robot*): *kittray* is in *robot*'s end effector.
- ☐ robot-holds-kittray(*robot*,*kittray*): *robot* is holding *kittray*.

2. put-kittray(*robot*,*kittray*,*worktable*): The *Robot robot* puts the *KitTray kittray* on the *WorkTable worktable*.

preconditions	*effects*
kittray-location-robot(*kittray*,*robot*)	¬kittray-location-robot(*kittray*,*robot*)
robot-holds-kittray(*robot*,*kittray*)	¬robot-holds-kittray(*robot*,*kittray*)
worktable-empty(*worktable*)	¬worktable-empty(*worktable*)
	kittray-location-worktable(*kittray*,*worktable*)
	robot-empty(*robot*)
	on-worktable-kittray(*worktable*,*kittray*)

- ■ *preconditions*
  - ☐ kittray-location-robot(*kittray*,*robot*): *kittray* is in *robot*'s end effector.
  - ☐ robot-holds-kittray(*robot*,*kittray*): *robot* holds *kittray*.
  - ☐ worktable-empty(*worktable*): There is nothing on *worktable*.
- ■ *effects*
  - ☐ ¬kittray-location-robot(*kittray*,*robot*): *kittray* is no longer it *robot*'s end effector since it is placed on *worktable*.
  - ☐ ¬robot-holds-kittray(*robot*,*kittray*): *robot* is not holding *kittray* anymore.
  - ☐ ¬worktable-empty(*worktable*): *worktable* is not empty anymore since there is something on top of it.
  - ☐ kittray-location-worktable(*kittray*,*worktable*): *kittray* is on *worktable*.
  - ☐ robot-empty(*robot*): *robot* is not holding anything.
  - ☐ on-worktable-kittray(*worktable*,*kittray*): *worktable* has *kittray* on top of it.

3. take-kit(*robot*,*kit*,*worktable*,*endeffector*): The *Robot robot* equipped with the *EndEffector endeffector* picks up the *Kit kit* from the *WorkTable worktable*.

preconditions	*effects*
kit-location-worktable(kit,$worktable$)	¬kit-location-worktable(kit,$worktable$)
robot-empty($robot$)	¬robot-empty($robot$)
on-worktable-kit($worktable$,kit)	¬on-worktable-kit($worktable$,kit)
robot-with-endeffector($robot$,$endeffector$)	kit-location-robot(kit,$robot$)
endeffector-type-kit($endeffector$,kit)	robot-holds-kit($robot$,kit)
	worktable-empty($worktable$)

- *preconditions*
  - ☐ kit-location-worktable(kit,$worktable$): kit is located on $worktable$.
  - ☐ robot-empty($robot$): $robot$ is not holding any object.
  - ☐ on-worktable-kit($worktable$,kit): $worktable$ has kit on top of it.
  - ☐ robot-with-endeffector($robot$,$endeffector$): $robot$ is equipped with $endeffector$.
  - ☐ endeffector-type-kit($endeffector$,kit): The type of $endeffector$ is capable of handling kit.
- *effects*
  - ☐ ¬kit-location-worktable(kit,$worktable$): kit is not on $worktable$.
  - ☐ ¬robot-empty($robot$): $robot$ is holding an object (kit).
  - ☐ ¬on-worktable-kit($worktable$,kit): $worktable$ does not have kit on top of it.
  - ☐ kit-location-robot(kit,$robot$): kit is being held by $robot$.
  - ☐ robot-holds-kit($robot$,kit): $robot$ is holding kit.
  - ☐ worktable-empty($worktable$): $worktable$ does not have any object on top of it.

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

preconditions	*effects*
kit-location-robot(kit,$robot$)	¬kit-location-robot(kit,$robot$)
robot-holds-kit($robot$,kit)	¬robot-holds-kit($robot$,kit)
lbwk-not-full($lbwk$)	kit-location-lbwk(kit,$lbwk$)
	robot-empty($robot$)

- *preconditions*
  - ☐ kit-location-robot(kit,$robot$): kit is held by $robot$.
  - ☐ robot-holds-kit($robot$,kit): $robot$ is holding kit.
  - ☐ lbwk-not-full($lbwk$): $lbwk$ should not be full so it can contain kit.
- *effects*

- ☐ ¬kit-location-robot(*kit*,*robot*): *kit* is not being held by *robot*.
- ☐ ¬robot-holds-kit(*robot*,*kit*): *robot* is not holding *kit*.
- ☐ kit-location-lbwk(*kit*,*lbwk*): *kit* has been placed in *lbwk*.
- ☐ robot-empty(*robot*): *robot* is not holding anything (not holding *kit* anymore).

5. *look-for-part*(*robot*,*part*,*partstray*,*kit*,*worktable*,*endeffector*): A sensor looks for the *Part part* in the *PartTray partstray*.

preconditions	*effects*
part-not-searched	¬part-not-searched
robot-empty(*robot*)	found-part(*partstray*)
robot-with-endeffector(*robot*,*endeffector*)	
on-worktable-kit(*worktable*,*kit*)	
endeffector-location-robot(*endeffector*,*robot*)	
part-location-partstray(*part*,*partstray*)	
kit-location-worktable(*kit*,*worktable*)	
endeffector-type-part(*endeffector*,*part*)	
partstray-not-empty(*partstray*)	

- ■ *preconditions*
  - ☐ part-not-searched: This flag is set to true in the initial state in the problem file (see section 3.2) and means that a part has not been searched yet.
  - ☐ robot-empty(*robot*): *robot* should not be holding anything. We want the operator *look-for-part* to be directly followed by the operator *take-part* to simulate a sensor identifying a part before being picked up by a robot. It is necessary to check that *robot*'s end effector is empty to prepare for the execution of the operator *take-part*.
  - ☐ robot-with-endeffector(*robot*,*endeffector*): *robot* is equipped with *endeffector*. Again, since we want the operator following *look-for-part* to be *take-part*, we want to make sure that *robot* is already equipped with *endeffector*.
  - ☐ on-worktable-kit(*worktable*,*kit*): *worktable* has *kit* on top of it.
  - ☐ endeffector-location-robot(*endeffector*,*robot*): *endeffector* is on *robot* so it is ready for the operator *take-part*.
  - ☐ part-location-partstray(*part*,*partstray*): *part* is in *partstray*.
  - ☐ kit-location-worktable(*kit*,*worktable*): *kit* is on *worktable*.
  - ☐ endeffector-type-part(*endeffector*,*part*): *endeffector* can handle *part*.
  - ☐ partstray-not-empty(*partstray*): *partstray* contains at least one part.
- ■ *effects*

- ☐ ¬part-not-searched: This flag is set to true so that the operator *look-for-part* can be called again to look for another part in the workstation.
- ☐ found-part(*partstray*): A part from *partstray* has been found.

6. *take-part*(*robot*,*part*,*partstray*,*endeffector*,*worktable*,*kit*): The *Robot* *robot* uses the *EndEffector* *endeffector* to pick up the *Part* *part* from the *PartTray* *partstray*.

preconditions	*effects*
part-location-partstray(*part*,*partstray*)	¬part-location-partstray(*part*,*partstray*)
robot-empty(*robot*)	¬robot-empty(*robot*)
endeffector-location-robot(*endeffector*,*robot*)	part-location-robot(*part*,*robot*)
robot-with-endeffector(*robot*,*endeffector*)	robot-holds-part(*robot*,*part*)
on-worktable-kit(*worktable*,*kit*)	decrease quantity-partstray(*partstray*)
kit-location-worktable(*kit*,*worktable*)	
endeffector-type-part(*endeffector*,*part*)	
partstray-not-empty(*partstray*)	
found-part(*part*,*partstray*)	

- ■ *preconditions*
  - ☐ part-location-partstray(*part*,*partstray*): *part* to be picked up is in *partstray*.
  - ☐ robot-empty(*robot*): *robot* is not holding any object.
  - ☐ endeffector-location-robot(*endeffector*,*robot*): *endeffector* is on *robot*.
  - ☐ robot-with-endeffector(*robot*,*endeffector*): *robot* is equipped with *endeffector*.
  - ☐ on-worktable-kit(*worktable*,*kit*): *worktable* has *kit* on top of it.
  - ☐ kit-location-worktable(*kit*,*worktable*): *kit* is on *worktable*. Once a part is picked up by the robot, the next logical action would be to put the part in the kit. For this to happen, the kit needs to be already on the work table so it can hold the part.
  - ☐ endeffector-type-part(*endeffector*,*part*): *endeffector* is the type for *part* handling.
  - ☐ partstray-not-empty(*partstray*): *partstray* is not empty and contains at least one part.
  - ☐ found-part(*part*,*partstray*): *part* has been found in *partstray*. found-part is set to true in the *effects* of the operator *look-for-part*.
- ■ *effects*
  - ☐ ¬part-location-partstray(*part*,*partstray*): *part* is not in *partstray* anymore since it was picked up by *robot*.

- ☐ ¬robot-empty(*robot*): *robot* is now holding *part* and is not empty anymore.
- ☐ part-location-robot(*part*,*robot*): *part* is held by *robot*.
- ☐ robot-holds-part(*robot*,*part*): *robot* is holding *part*.
- ☐ decrease quantity-partstray(*partstray*): After picking up a part from *partstray* the number of parts in *partstray* is decreased by one. This is expressed with the decrease function.

7. put-part(*robot*,*part*,*kit*,*worktable*,*partstray*): The *Robot robot* puts the *Part part* in the *Kit kit*.

preconditions	*effects*
part-location-robot(*part*,*robot*)	¬part-location-robot(*part*,*robot*)
robot-holds-part(*robot*,*part*)	¬robot-holds-part(*robot*,*part*)
on-worktable-kit(*worktable*,*kit*)	¬found-part(*part*,*partstray*)
kit-location-worktable(*kit*,*worktable*)	robot-empty(*robot*)
origin-part(*part*,*partstray*)	part-location(*part*,*kit*)
(< (quantity-kit(*kit*,*partstray*)) (capacity-kit(*kit*,*partstray*)))	(increase (quantity-kit(*kit*,*partstray*)))
	part-not-searched

- ■ *preconditions*
  - ☐ part-location-robot(*part*,*robot*): *part* is held by *robot*.
  - ☐ robot-holds-part(*robot*,*part*): *robot* is holding *part*.
  - ☐ on-worktable-kit(*worktable*,*kit*): *worktable* has *kit* on top of it.
  - ☐ kit-location-worktable(*kit*,*worktable*): *kit* is on *worktable*.
  - ☐ origin-part(*part*,*partstray*): *part* is from *partstray*. This is used to tell the type of *part*.
  - ☐ (< (quantity-kit(*kit*,*partstray*)) (capacity-kit(*kit*,*partstray*))): The quantity of parts of type *partstray* in *kit* should be lesser than the capacity *kit* can hold for this type of part.
- ■ *effects*
  - ☐ ¬part-location-robot(*part*,*robot*): *part* is not held by *robot*.
  - ☐ ¬robot-holds-part(*robot*,*part*): *robot* is not holding *part*.
  - ☐ robot-empty(*robot*): *robot* is not holding any object.
  - ☐ part-location(*part*,*kit*): *part* is located in *kit*.
  - ☐ (increase (quantity-kit(*kit*,*partstray*))): Once *part* is placed in *kit*, the quantity of *parts* in *kit* is increased by one.
  - ☐ part-not-searched: This flag is set to true so another part search (through the operator look-for-part) is made after *part* is placed in *kit*.

8. attach-endeffector(*robot*,*endeffector*,*endeffectorholder*,*endeffectorchangingstation*): The *Robot robot* attaches the *EndEffector endeffector* which is situated in the *EndEffectorHolder endeffectorholder*.

preconditions
endeffector-location-endeffectorholder(*endeffector*,*endeffectorholder*)
robot-with-no-endeffector(*robot*)
endeffectorholder-holds-endeffector(*endeffectorholder*,*endeffector*)
endeffectorholder-location(*endeffectorholder*,*endeffectorchangingstation*)
endeffectorchangingstation-contains-endeffectorholder(*endeffectorchangingstation*,*endeffectorholder*)

effects
¬endeffector-location-endeffectorholder(*endeffector*,*endeffectorholder*)
¬endeffectorholder-holds-endeffector(*endeffectorholder*,*endeffector*)
¬robot-with-no-endeffector(*robot*)
robot-empty(*robot*)
endeffector-location-robot(*endeffector*,*robot*)
robot-with-endeffector(*robot*,*endeffector*)
endeffectorholder-empty(*endeffectorholder*)

- *preconditions*
  - endeffector-location-endeffectorholder(*endeffector*,*endeffectorholder*): *endeffector* is located in *endeffectorholder*.
  - robot-with-no-endeffector(*robot*): *robot* is not equipped with any *endeffector*.
  - endeffectorholder-holds-endeffector(*endeffectorholder*,*endeffector*): *endeffectorholder* is holding *endeffector*.
  - endeffectorholder-location(*endeffectorholder*,*endeffectorchangingstation*): *endeffectorholder* is in *endeffectorchangingstation*.
  - endeffectorchangingstation-contains-endeffectorholder(*endeffectorchangingstation*,*endeffecto* *endeffectorchangingstation* contains *endeffectorholder*.
- *effects*
  - ¬endeffector-location-endeffectorholder(*endeffector*,*endeffectorholder*): *endeffector* is not in *endeffectorholder* anymore since it has been attached to *robot*.
  - ¬endeffectorholder-holds-endeffector(*endeffectorholder*,*endeffector*): *endeffectorholder* is not holding *endeffector* anymore.
  - ¬robot-with-no-endeffector(*robot*): *robot* is now equipped with *endeffector*.
  - robot-empty(*robot*): *robot* is not holding any object.
  - endeffector-location-robot(*endeffector*,*robot*): *endeffector* is on *robot*.

- □ robot-with-endeffector(*robot*,*endeffector*): *robot* is equipped with *endeffector*.
- □ endeffectorholder-empty(*endeffectorholder*): *endeffectorholder* is not holding any *endeffector*.

9. remove-endeffector(*robot*,*endeffector*,*endeffectorholder*,*endeffectorchangingstation*): The *Robot robot* removes the *EndEffector endeffector* and puts it in the *EndEffectorHolder endeffectorholder*.

preconditions
endeffector-location-robot(*endeffector*,*robot*)
robot-with-endeffector(*robot*,*endeffector*)
robot-empty(*robot*)
endeffectorholder-location(*endeffectorholder*,*endeffectorchangingstation*)
endeffectorchangingstation-contains-endeffectorholder(*endeffectorchangingstation*,*endeffectorholder*)
endeffectorholder-empty(*endeffectorholder*)

effects
¬endeffector-location-robot(*endeffector*,*robot*)
¬robot-with-endeffector(*robot*,*endeffector*)
¬endeffectorholder-empty(*endeffectorholder*)
endeffector-location-endeffectorholder(*endeffector*,*endeffectorholder*)
endeffectorholder-holds-endeffector(*endeffectorholder*,*endeffector*)
robot-with-no-endeffector(*robot*)

- ■ *preconditions*
  - □ endeffector-location-robot(*endeffector*,*robot*): *endeffector* is on *robot*.
  - □ robot-with-endeffector(*robot*,*endeffector*): *robot* is holding *endeffector*.
  - □ robot-empty(*robot*): *robot* is not holding anything.
  - □ endeffectorholder-location(*endeffectorholder*,*endeffectorchangingstation*): *endeffectorholder* is in *endeffectorchangingstation*.
  - □ endeffectorchangingstation-contains-endeffectorholder(*endeffectorchangingstation*,*endeffectorholder*)
  - □ endeffectorholder-empty(*endeffectorholder*): *endeffectorholder* does not contain *endeffector*.
- ■ *effects*
  - □ ¬endeffector-location-robot(*endeffector*,*robot*): *endeffector* is not on *robot* anymore.
  - □ ¬robot-with-endeffector(*robot*,*endeffector*): *robot* does not have *endeffector* anymore.

- ☐ ¬endeffectorholder-empty(*endeffectorholder*): *endeffectorholder* does not contain any *endeffector*.
- ☐ endeffector-location-endeffectorholder(*endeffector*,*endeffectorholder*): *endeffector* is situated in *endeffectorholder*.
- ☐ endeffectorholder-holds-endeffector(*endeffectorholder*,*endeffector*): *endeffectorholder* holds *endeffector*.
- ☐ robot-with-no-endeffector(*robot*): *robot* is not equipped with *endeffector*.

10. *create-kit*(*kit*,*kittray*,*worktable*): The *KitTray kittray* is converted into the *Kit kit* once the *KitTray kittray* is on the *WorkTable worktable*.

preconditions	*effects*
on-worktable-kittray(*worktable*,*kittray*)	¬on-worktable-kittray(*worktable*,*kittray*)
	on-worktable-kit(*worktable*,*kit*)
	kit-location-worktable(*kit*,*worktable*)

- ■ *preconditions*
  - ☐ on-worktable-kittray(*worktable*,*kittray*): *worktable* has *kittray* on top of it.
- ■ *effects*
  - ☐ ¬on-worktable-kittray(*worktable*,*kittray*): The object *kittray* is destroyed and is thus not on *worktable* anymore.
  - ☐ on-worktable-kit(*worktable*,*kit*): *worktable* now has *kit* on top of it.
  - ☐ kit-location-worktable(*kit*,*worktable*): *kit* is on *worktable*.

2.5.2 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-part*(*robot*,*part*,*partstray*,*endeffector*) in the kitting domain can be translated into the action *take-part*(*robot_1*,*part_1*,*partstray_1*,*endeffector_2*) where *robot_1*, *part_1*, *partstray_1*, and *endeffector_2* are constant variable symbols in the classes *Robot*, *Part*, *PartsTray*, and *EndEffector*, respectively.

3 Planning Language

The Planning Domain Definition Language (PDDL) [3] 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.

3.1 The PDDL Domain File

The PDDL domain file is composed of four sections that include requirements, types, predicates and functions, and actions. An excerpt of the PDDL domain file is depicted in Figure 1.

- line 1: The keyword `domain` signals a planner that this file contains information on the domain. `kitting-domain` is the name given to the domain.
- line 2: The `:requirements` field specifies which section the domain relies on. The planning system can examine this statement to determine if it is capable of solving problems in this domain. A keyword (symbol starting with a colon) used in a `:requirements` field is called a requirement flag; the domain is said to declare a requirement for that flag. The requirement flags present in the kitting domain are:
    - `:strips`: The most basic subset of PDDL, consisting of STRIPS only.
    - `:typing`: PDDL has a special syntax for declaring parameter and object types. `:typing` allows types names in declaration of variables.
    - `:fluents`: A domain's set of requirements allow a planner to quickly tell if it is likely to be able to handle the domain. For example, this version of the kitting world requires fluents numeric,

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

    (:predicates
           (endeffector-location-robot ?endeffector - EndEffector ?robot - Robot)
           (on-worktable-kit ?worktable - WorkTable ?kit - Kit)
    )

    (:functions
           (quantity-partstray ?partstray - PartsTray)
           (quantity-kit ?kit - Kit ?partstray - PartsTray)
           (capacity-kit ?kit - Kit ?partstray - PartsTray)
    )

    (:action take-kittray
        :parameters(
            ?robot - Robot
            ?kittray - KitTray
            ?largeboxwithemptykittrays - LargeBoxWithEmptyKitTrays
            ?endeffector - EndEffector
            ?worktable - WorkTable)
        :precondition(and
            (robot-empty ?robot)
            (lbwekt-not-empty ?largeboxwithemptykittrays)
            (robot-with-endeffector ?robot ?endeffector)
            (kittray-location-lbwekt ?kittray ?largeboxwithemptykittrays)
            (endeffector-location-robot ?endeffector ?robot)
            (worktable-empty ?worktable)
            (endeffector-type-kittray ?endeffector ?kittray))
        :effect(and
            (robot-holds-kittray ?robot ?kittray)
            (kittray-location-robot ?kittray ?robot)
            (not (robot-empty ?robot))
            (not (kittray-location-lbwekt ?kittray ?largeboxwithemptykittrays)))
    )
)
```

Figure 1: Excerpt of the PDDL domain file for kitting.

so a straight STRIPS-representation planner would not be able to handle it. A fluent is a term (`:functions`) with time-varying value (i.e., a value that can change as a result of performing an action).

- line 3–15: Type names have to be declared before they are used (before `:predicates` and `:functions`). This is done with the declaration `(:types` $name_1$ `...` $name_n$`)`.

- line 17–20: The `:predicates` part of a domain definition specify only what are the predicate names used in the domain, and their number of arguments (and argument types, if the domain uses `:typing`). The "meaning" of a predicate, in the sense of for what combinations of arguments it can be true and its relationship to other predicates, is determined by the effects that actions in the domain can have on the predicate, and by what instances of the predicate are listed as true in the initial state of the problem definition.
  It is common to make a distinction between static and dynamic predicates. A *static* predicate is not changed by any action. Thus in a problem, the true and false instances of a *static* predicate will always be precisely those listed in the initial state specification of the problem definition. Note that there is no syntactic difference between *static* and *dynamic* predicates in PDDL, they look exactly the same in the `:predicates` declaration part of the domain.
  A predicate is build using the structure `(predicate_name ?X - type_of_X)`. A list of parameters of the same type in a predicate can be abbreviated to `(predicate_name ?X ?Y ?Z - type_of_XYZ)`. Note that the hyphen between parameter and type name is surrounded by whitespace.

- line 22–26: A fluent is similar to a state variable/predicate except that its value is a number instead of true or false. The initial value of a function is set in the initial state of the problem file and changes when an action is executed. The declaration of functions is similar to predicates.

- line 28–48: The domain definition contains operators (called *actions* in PDDL). An action statement specifies a way that a planner affects the state of the world. The statement includes parameters, preconditions, and effects. All parts of an action definition except the name are, according to the PDDL specification, optional (although, of course, an action without effects is pretty useless). However, for an action that has no preconditions some planners may require an "empty" precondition, on the form :precondition () or :precondition (and), and some planners may also require an empty :parameter list for actions without parameters).

- ☐ line 29–34: The `:parameters` section declare all the parameters used by predicates and functions in `preconditions` and `effects`.
- ☐ line 35–42: The `:preconditions` section is a conjunction of predicates and functions that need to be true in the world in order for the action to be invoked.
- ☐ line 43–47: The `:effects` equation dictates the changes in the world that will occur due to the execution of the action.

3.2 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). Using an example of kit to build, this section only describes the initial and goal states explicitly. The operators detailed in Section 2.5 are used by a planner to generate the other states as needed.
In this example, the *Robot* has to build a kit that contains two *Parts* of type A, two *Part* of type B and one *Part* of type C. The kitting process is completed once the *Kit* is placed in the *LargeBoxWithKits*. The PDDL problem file for the kitting domain is presented below.

```
(define (problem kitting-problem)
    (:domain kitting-domain)
    (:objects
        robot_1 - Robot
        changing_station_1 - EndEffectorChangingStation
        kit_tray_1 - KitTray
        kit_a2b2c1 - 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_a_3 part_a_4 - Part
        part_b_1 part_b_2 part_b_3 part_b_4 - Part
        part_c_1 part_c_2 part_c_3 part_c_4 - Part
        part_gripper tray_gripper - EndEffector
        part_gripper_holder tray_gripper_holder - EndEffectorHolder
    )
)
(:init
    (robot-with-no-endeffector robot_1)
    (part-not-searched)
    (lbwekt-not-empty empty_kit_tray_supply)
    (lbwk-not-full finished_kit_receiver)
    (partstray-not-empty part_a_tray)
    (partstray-not-empty part_b_tray)
    (partstray-not-empty part_c_tray)
    (endeffector-location-endeffectorholder part_gripper part_gripper_holder)
    (endeffector-location-endeffectorholder tray_gripper tray_gripper_holder)
    (endeffectorholder-holds-endeffector part_gripper_holder part_gripper)
    (endeffectorholder-holds-endeffector tray_gripper_holder tray_gripper)
    (endeffectorholder-location tray_gripper_holder changing_station_1)
    (endeffectorholder-location part_gripper_holder changing_station_1)
    (endeffectorchangingstation-contains-endeffectorholder changing_station_1 tray_gripper_holder)
    (endeffectorchangingstation-contains-endeffectorholder changing_station_1 part_gripper_holder)
    (worktable-empty work_table_1)
    (kittray-location-lbwekt kit_tray_1 empty_kit_tray_supply)

    (part-location-partstray part_a_1 part_a_tray)
    (part-location-partstray part_a_2 part_a_tray)
    (part-location-partstray part_a_3 part_a_tray)
    (part-location-partstray part_a_4 part_a_tray)
    (part-location-partstray part_b_1 part_b_tray)
    (part-location-partstray part_b_2 part_b_tray)
    (part-location-partstray part_b_3 part_b_tray)
    (part-location-partstray part_b_4 part_b_tray)
    (part-location-partstray part_c_1 part_c_tray)
    (part-location-partstray part_c_2 part_c_tray)
    (part-location-partstray part_c_3 part_c_tray)
    (part-location-partstray part_c_4 part_c_tray)

    (endeffector-type-part part_gripper part_a_1)
    (endeffector-type-part part_gripper part_a_2)
    (endeffector-type-part part_gripper part_a_3)
    (endeffector-type-part part_gripper part_a_4)
    (endeffector-type-part part_gripper part_b_1)
    (endeffector-type-part part_gripper part_b_2)
    (endeffector-type-part part_gripper part_b_3)
    (endeffector-type-part part_gripper part_b_4)
    (endeffector-type-part part_gripper part_c_1)
    (endeffector-type-part part_gripper part_c_2)
    (endeffector-type-part part_gripper part_c_3)
    (endeffector-type-part part_gripper part_c_4)
    (endeffector-type-kittray tray_gripper kit_tray_1)
    (endeffector-type-kit tray_gripper kit_a2b2c1)
```

```
65.     (= (capacity-kit kit_a2b2c1 part_a_tray) 2)
66.     (= (capacity-kit kit_a2b2c1 part_b_tray) 2)
67.     (= (capacity-kit kit_a2b2c1 part_c_tray) 1)
68.     (= (quantity-kit kit_a2b2c1 part_a_tray) 0)
69.     (= (quantity-kit kit_a2b2c1 part_b_tray) 0)
70.     (= (quantity-kit kit_a2b2c1 part_c_tray) 0)
71.     (= (quantity-partstray part_a_tray) 4)
72.     (= (quantity-partstray part_b_tray) 4)
73.     (= (quantity-partstray part_c_tray) 4)
74.
75.     (origin-part part_a_1 part_a_tray)
76.     (origin-part part_a_2 part_a_tray)
77.     (origin-part part_a_3 part_a_tray)
78.     (origin-part part_a_4 part_a_tray)
79.     (origin-part part_b_1 part_b_tray)
80.     (origin-part part_b_2 part_b_tray)
81.     (origin-part part_b_3 part_b_tray)
82.     (origin-part part_b_4 part_b_tray)
83.     (origin-part part_c_1 part_c_tray)
84.     (origin-part part_c_2 part_c_tray)
85.     (origin-part part_c_3 part_c_tray)
86.     (origin-part part_c_4 part_c_tray)
87. )
88.
89. (:goal
90.     (and
91.         (= (quantity-kit kit_a2b2c1 part_a_tray) (capacity-kit kit_a2b2c1 part_a_tray))
92.         (= (quantity-kit kit_a2b2c1 part_b_tray) (capacity-kit kit_a2b2c1 part_b_tray))
93.         (= (quantity-kit kit_a2b2c1 part_c_tray) (capacity-kit kit_a2b2c1 part_c_tray))
94.         (kit-location-lbwk kit_a2b2c1 finished_kit_receiver)
95.     )
96. )
```

- line 1: Signal a planner that the file contains all the element part of a problem. `kitting-problem` is the name given to this problem.

- line 2: `:domain` refers to the domain that the current problem is associated to. In this case, the problem refers to the domain `kitting-domain`. Note that `kitting-domain` is the name given to the kitting domain as presented in section 3.1.

- line 3–17: `:objects` declare objects present in the problem instance. The syntax for `:objects` is $object_1$ `- Type ...` $object_n$ `- Type`.

3.2.1 Initial State

The initial state S_0 (Figure 2) defines the environment in its initial condition. The initial state of the kitting problem in PDDL format is described below.

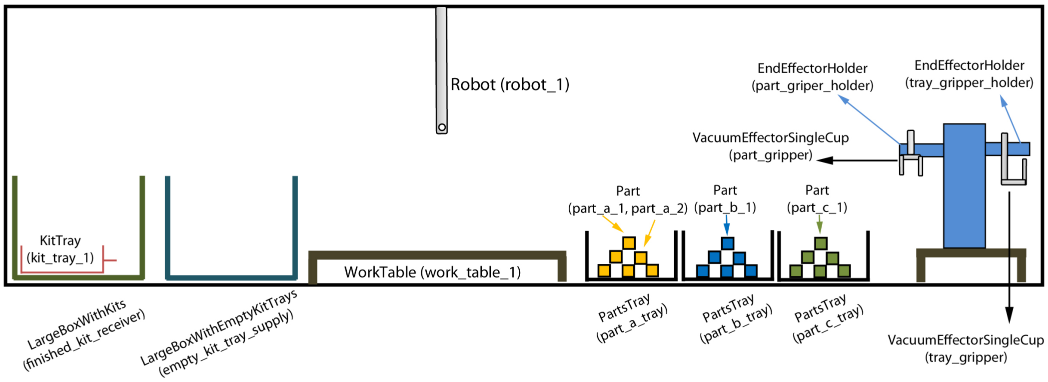

Figure 2: Initial state S_0.

- line 19: `:init` signals a planner that the predicates and functions in this section are true in the initial state.

- line 20–87: Predicates true in the initial state of the environment. Since PDDL uses a close world assumption, predicates that are not present in the initial state are automatically set to false. This section also set the initial values for functions. Some relevant sections are presented:

  - ☐ line 21: The predicate part-not-searched is set to true so that the operator *look-for-part* can be activated during a plan search.
  - ☐ line 65–67: Functions describing the quantity of parts of a type that *kit_a2b2c1* can contain. In this example, *kit_a2b2c1* can have 2 parts of type A (*part_a_tray*), 2 parts of type B (*part_b_tray*), and 1 part of type C (*part_c_tray*).
  - ☐ line 68–70: Functions that represent the quantity of parts of a specific type that are already in *kit_a2b2c1*. *kit_a2b2c1* has no parts of type A, B, and C.
  - ☐ line 71–73: Functions that describe the quantity of parts available in their respective parts tray. This also can be read as: *In the workstation, there are 4 parts of type A available, 4 parts of type B available, and 4 parts of type C available.*
  - ☐ line 75–86: Predicates that describe the type of each specific part in the workstation.

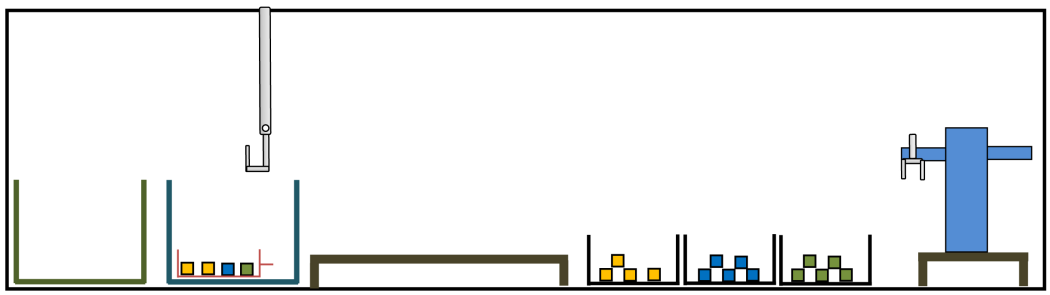

Figure 3: Goal state S_G.

3.2.2 Goal State

Figure 3 depicts the goal state S_G for the kitting workstation, followed by a representation of the goal state in PDDL format.

- line 89: `:goal` is a keyword used to signal a planner about the goal state to reach. All the predicates and functions in the goal state must be true.
- line 91–93: The quantity of parts of a specific type in *kit_a2b2c1* should match the capacity of parts of a specific type for *kit_a2b2c1*. The quantity of parts in *kit_a2b2c1* is increased in the operator put-part. The initial quantity of parts in *kit_a2b2c1* and its capacity are set in the initial state. Note that we are not specifying which instance of *Part* should go in *kit_a2b2c1* but rather the number of *Parts* of a specific type that *kit_a2b2c1* must have.
- line 94: *kit_a2b2c1* should be placed in the large box with kits *finished_kit_receiver*.

3.3 Plan

This section shows an example of plan generated by the planner described in section 4. Figure 4 displays the different states and actions used by the planner to generate a plan starting from the initial state S_0 to the goal state S_G. The actions $A_1 \ldots A_{17}$ are described below.

- *A*1:(attach-endeffector *robot_1* *tray_gripper* *tray_gripper_holder* *changing_station_1*)

- *A2*:(take-kittray *robot_1 kit_tray_1 empty_kit_tray_supply tray_gripper work_table_1*)
- *A3*:(put-kittray *robot_1 kit_tray_1 work_table_1*)
- *A4*:(create-kit *kit_a2b2c1 kit_tray_1 work_table_1*)
- *A5*:(remove-endeffector *robot_1 tray_gripper tray_gripper_holder changing_station_1*)
- *A6*:(attach-endeffector *robot_1 part_gripper part_gripper_holder changing_station_1*)
- *A7*:(look-for-part *robot_1 part_c_1 part_c_tray kit_a2b2c1 work_table_1 part_gripper*)
- *A8*:(take-part *robot_1 part_c_1 part_c_tray part_gripper work_table_1 kit_a2b2c1*)
- *A9*:(put-part *robot_1 part_c_1 kit_a2b2c1 work_table_1 part_c_tray*)
- *A10*:(look-for-part *robot_1 part_b_2 part_b_tray kit_a2b2c1 work_table_1 part_gripper*)
- *A11*:(take-part *robot_1 part_b_2 part_b_tray part_gripper work_table_1 kit_a2b2c1*)
- *A12*:(put-part *robot_1 part_b_2 kit_a2b2c1 work_table_1 part_b_tray*)
- *A13*:(look-for-part *robot_1 part_b_1 part_b_tray kit_a2b2c1 work_table_1 part_gripper*)
- *A14*:(take-part *robot_1 part_b_1 part_b_tray part_gripper work_table_1 kit_a2b2c1*)
- *A15*:(put-part *robot_1 part_b_1 kit_a2b2c1 work_table_1 part_b_tray*)
- *A16*:(look-for-part *robot_1 part_a_2 part_a_tray kit_a2b2c1 work_table_1 part_gripper*)
- *A17*:(take-part *robot_1 part_a_2 part_a_tray part_gripper work_table_1 kit_a2b2c1*)
- *A18*:(put-part *robot_1 part_a_2 kit_a2b2c1 work_table_1 part_a_tray*)
- *A19*:(look-for-part *robot_1 part_a_1 part_a_tray kit_a2b2c1 work_table_1 part_gripper*)
- *A20*:(take-part *robot_1 part_a_1 part_a_tray part_gripper work_table_1 kit_a2b2c1*)

- *A21*:(put-part *robot_1 part_a_1 kit_a2b2c1 work_table_1 part_a_tray*)
- *A22*:(remove-endeffector *robot_1 part_gripper part_gripper_holder changing_station_1*)
- *A23*:(attach-endeffector *robot_1 tray_gripper tray_gripper_holder changing_station_1*)
- *A24*:(take-kit *robot_1 kit_a2b2c1 work_table_1 tray_gripper*)
- *A25*:(put-kit *robot_1 kit_a2b2c1 finished_kit_receiver*)

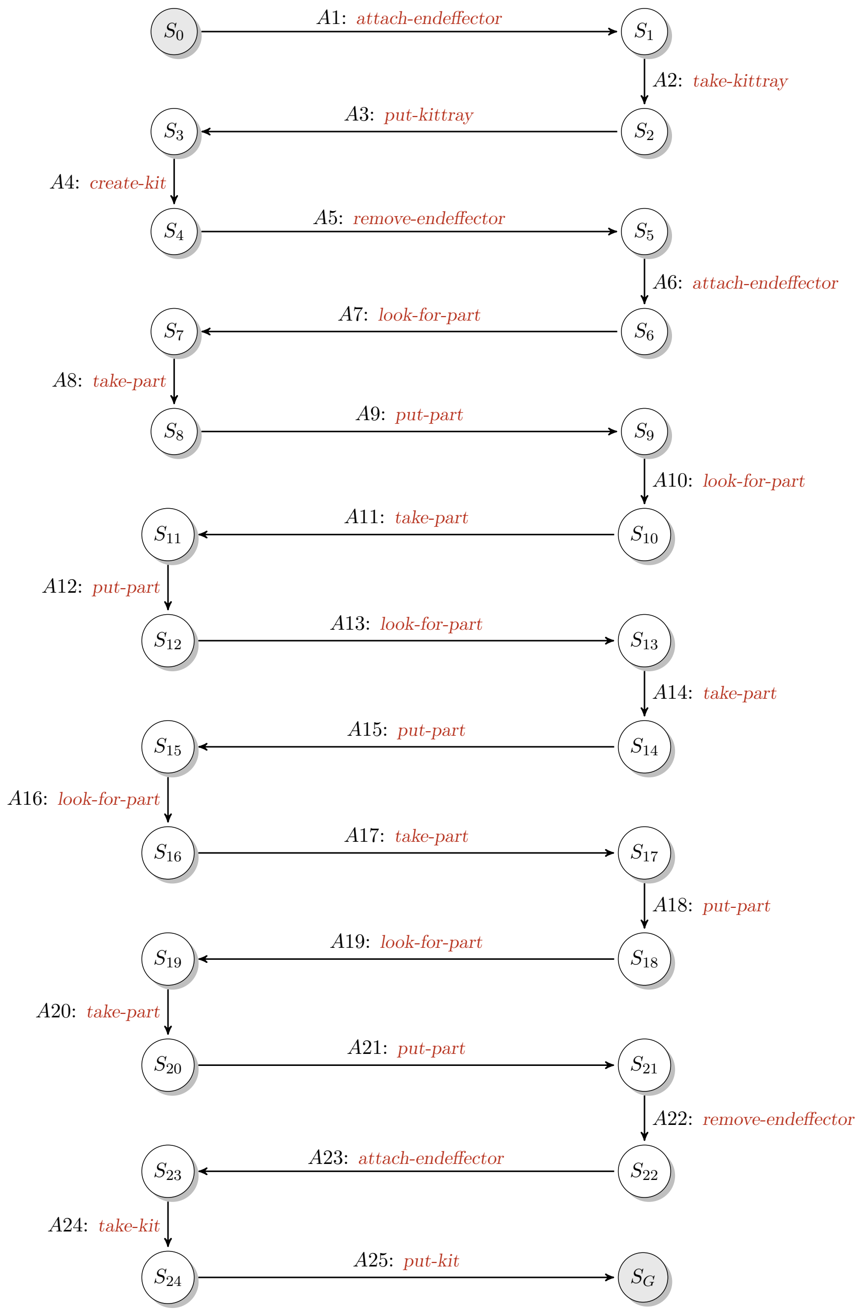

Figure 4: Example of plan generated with the kitting domain and problem files.

4 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 uses a forward-chaining partial-order planning [1].

4.1 Requirements

The planner requires:

- `cmake`
- The CBC mixed integer programming solver (`https://projects.coin-or.org/Cbc/`)
- `perl`, `bison` and `flex` to build the parser

These are packaged with most Linux distributions - on Ubuntu/Debian, the following should suffice:

```
sudo apt-get install cmake coinor-libcbc-dev coinor-libclp-dev
\
coinor-libcoinutils-dev bison flex
```

4.2 Download and Install CBC

The CBC source code can be obtained using subversion:

- `svn co https://projects.coin-or.org/svn/Cbc/stable/2.7 coin-Cbc`: Issues the subversion command to obtain the source code.
- `cd coin-Cbc`
- `./configure -C`: Runs a configure script that generates the make file.
- `make`: Builds the Cbc library and executable program.
- `make test`: Builds and runs the Cbc unit test program.
- `make install`: Installs libraries, executables and header files in directories `coin-Cbc/lib`, `coin-Cbc/bin` and `coin-Cbc/include`.

4.3 Compile the Planner

Before compiling the planner, the file *compile/CMakeCache.txt* should be edited as follows. Note that `<path>` is the absolute path that leads to the `coin-Cbc` directory.

- `CBC_INCLUDES:PATH = <path>/coin-Cbc/build/include`
- `CGL_INCLUDES:PATH = <path>/coin-Cbc/build/include`
- `CLP_INCLUDES:PATH = <path>/coin-Cbc/build/include`
- `COINUTILS_INCLUDES:PATH = <path>/coin-Cbc/build/include/coin`
- `OSI_INCLUDES:PATH = <path>/coin-Cbc/build/include`

To compile the planner, one should use:

```
./build
```

4.4 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 following command run the planner on the PDDL files.

```
./plan <domain> <problem> <solution>
```

Where `<domain>` and `<problem>` are the PDDL domain and problem files, respectively.`<solution>` is the output file containing the plan.

5 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.

5.1 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.

5.1.1 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`.

5.1.2 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*

5.2 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 Open-JDK Java 6 Runtime". Note that this message will be different for future releases of the Java Runtime Environment.

5.3 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.

5.3.1 OWL to SQL

To convert OWL classes and instances to SQL, the `Owl to SQL` tab should be selected (see Figure 5). The different fields are:

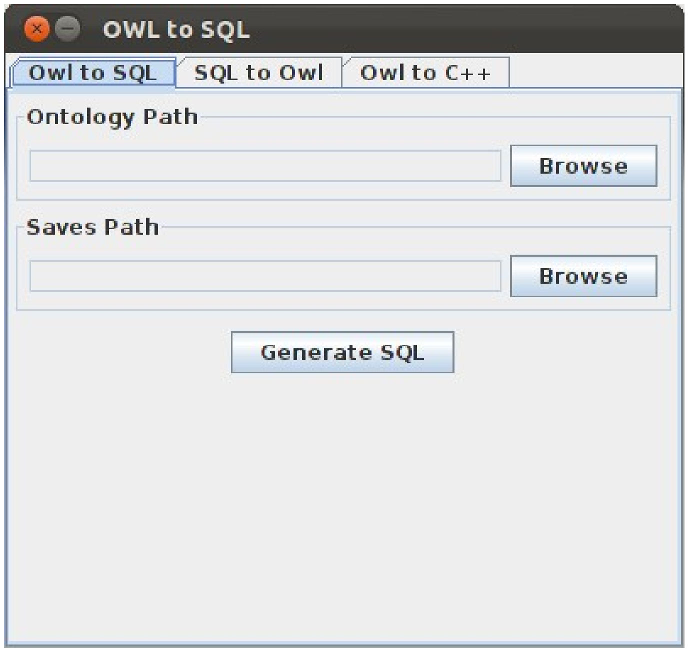

Figure 5: Owl to SQL tab.

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.

5.3.2 OWL to C++

The "Owl to C++" tab (see Figure 6) 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.

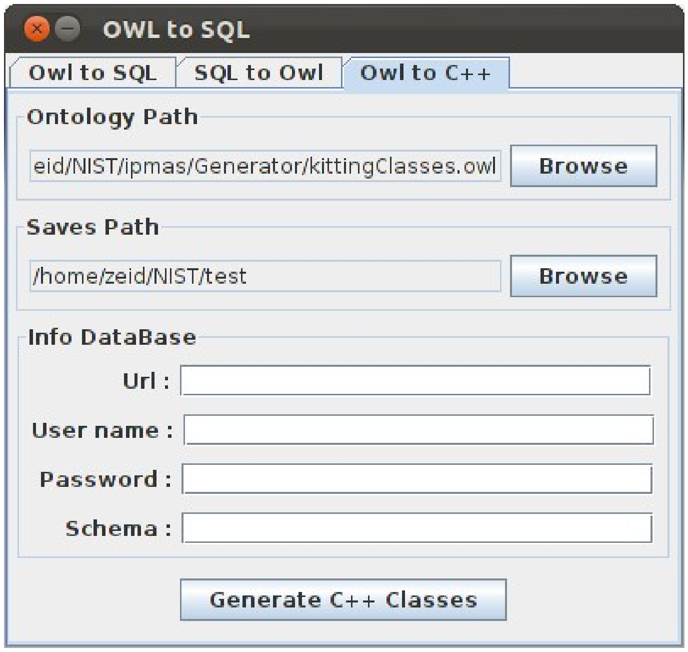

Figure 6: Owl to C++ tab.

- **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.
- **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.

References

[1] A. J. Coles, A. Coles, M. Fox, and D. Long. Forward-Chaining Partial-Order Planning. In *20th International Conference on Automated Planning and Scheduling, ICAPS 2010*, pages 42–49, Toronto, Ontario, Canada, May 12–16 2010. AAAI 2010.

[2] M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research (JAIR)*, 20(1):61–124, 2003.

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

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

List of Figures