The Manufacturing Plant Domain

J. Benton * and **Minh B. Do** † and **Wheeler Ruml** †
* CSE Department Arizona State Univ. Tempe, AZ 85287, USA, j.benton@asu.edu
† Embedded Reasoning Area, Palo Alto Research Center, Palo Alto, CA 94304, USA, {minhdo,ruml}@parc.com

The manufacturing plant domain consists of a set of machines linked together in a directed planar graph G. Additionally, the domain includes a set of input and output devices. Input devices must have only out-links and output devices must have only in-links. Input and output devices must only link to machines on the convex hull of G. Product is passed from input devices to output devices, both of which have infinite capacity. When product exists inside of a machine, that machine may give some attribute to the product. Only one product may exist at a given time in most machines (with the only exception being that more than one may exist in input and output devices). Goals consist of (1) the initial location of the product (an input device) (2) the final destination of the product (an output device) and (3) the set of attributes which the product must possess. A version of this domain was originally introduced in (Ruml, Do, & Fromherz 2005).

The domain is written in the standard STRIPS-like PDDL 2.1 format. It uses simple durative actions (i.e., action template definitions with constant duration) and contains no numeric functions.

Objects

Three object types exist in the domain. They include:

- **machine** - Machines transport product and give attributes to it.
- **product** - Objects of this type can be transported through the machine. Product is delivered from an input node to an output node.
- **attribute** - An attribute α can be applied to product by machines that give α.

Actions

The domain consists of three actions which are listed in Figure 1. The action *transport* moves product through the manufacturing plant between machines. A similar action, *transport-out*, moves product from a machine to an attached output device. The final action, *add-attribute*, gives attributes to product at a given machine.

Figures 2, 3 and 4 list the parameters for each of the actions. Both the *transport* and *transport-out* actions move product, taking parameters that indicate the product to be transported, the originating machine and the destination machine. These machines must be linked to one another. A machine is known to be linked given a predicate `(connected ?mach1 ?mach2)` where `mach1` and `mach2` can be any two machines. Transport to a machine is contingent upon whether the machine already contains some product. The exception to this rule is output devices. The `transport-out` action is a special action that transports to these nodes, which are specified with the predicate `(is-output ?mach)`.

The action *add-attribute* gives an attribute to product at a given machine. Each machine has a specified set of attributes that it may give. The *transport* and *transport-out* actions have a duration of 10 units of time and the *add-attribute* action has a duration of 2 units of time.

Action	Description
transport	Transport product between two machines.
transport-out	Transport product to an output node.
add-attribute	Add an attribute to the product.

Figure 1: Actions in the manufacturing plant domain.

Parameter Type	Description
product	Product that is being transported.
machine	The machine from which we are transporting the product.
machine	The specified machine that we are moving the product to. This cannot be an output device.

Figure 2: The parameters of the *transport* action.

Parameter Type	Description
product	Product that is being transported.
machine	The machine from which we are transporting the product.
machine	The specified output device that we are moving the product to.

Figure 3: The parameters of the *transport-out* action.

Parameter Type	Description
product	Material gaining the attribute.
machine	The machine giving the attribute.
attribute	The attribute that is being given to the product.

Figure 4: The parameters of the *add-attribute* action.

Goals

In this domain goals arrive on-line asynchronously and are not known by a planner *a priori*. Instead, the planner must handle these goals while keeping aware of resources that may be in use and unavailable from previously scheduled operations. The objective is to minimize the amount of time taken to achieve all of the on-line goals, which are defined as a conjunctive set that must become true at some future time. Since goals are not known by planners up-front, they must be sent at a time specified to the simulator (but not known to competitors). A particular set of goals will appear only once.

The goals consist of product with specified attributes arriving at an output device. They also include the initial location of the product involved. The predicates involved in goal achievement, `at` and `has-attribute` take the parameters given in Figures 5 and 6.

Parameter	Description
product	A specific product.
machine	The location of the product.

Figure 5: The parameters of the *at* predicate.

Parameter	Description
product	Product with the given attribute.
attribute	An attribute given by a machine.

Figure 6: The parameters of the *has-attribute* predicate.

Plan Format

The simulator accepts plans in a format similar to that used in the International Planning Competitions. That is:

```
<start-time>: <action>
```

Note that unlike the planning competition format, action durations need not be given. A plan in our domain might look like:

```
0.0: (transport product1 input1 machine1)
10.01: (add-attribute product1 machine1 attribute1)
12.02: (transport-out product1 machine1 output1)
```

Goal Arrival and Action Incompatibility

The simulation architecture for this domain is unique in that new goals continually arrive. As such, the tools developed to interact with this domain must be ready to receive new goals and send solutions to satisfy them. The simulator accepts actions (i.e., plans) at any time until all goals have been sent and satisfied. Also, because the simulation continues after new goals are sent, solutions entered into the simulator must not interfere with solutions already executing.[1]

Two actions interfere (i.e., are incompatible) with one another under the following conditions:

- A delete effect of a starting action is the same as a precondition of an executing action.
- A precondition of an starting action is the same as the delete effect of an executing action.
- A delete effect of a starting action is the same as an add effect of an executing action.
- An add effect of a starting action is the same as the delete effect of an executing action.

Additionally, for an action to execute, all of its preconditions must be satisfied. Both action incompatibility and unsatisfied preconditions cause the simulator to halt and a message to be logged indicating the reason of the halting condition.

Log

The objective of the simulation is to minimize goal achievement time and so the log consists of a list of goals together with the time points at which they are achieved.

Its format is as follows:

```
<goal>:<goal-achievement-time>
```

Goals are displayed in standard PDDL2.1 format as in:

```
(at product1 machine1): 2.3
```

Action interference is also logged. That is, the simulator will log when an action is sent that cannot execute due to constraints caused by another action. For instance, if an action `(transport product1 machine2 machine3)` is executing between time 1 and 11 and the action `(transport product2 machine1 machine2)` is to begin executing at time 3, the simulator will halt and an entry in the log will read[2]:

```
Action (transport product2 machine1 machine2) at 3.0
interferes with action
(transport product1 machine2 machine3) at 1.0
```

Additionally, the simulator logs when an action's preconditions fail to be met at its specified execution time as follows:

```
Action (transport product2 machine1 machine2)
preconditions not met at 2.0
```

References

Ruml, W.; Do, M. B.; and Fromherz, M. P. J. 2005. On-line planning and scheduling for high-speed manufacturing. In *ICAPS*, 30–39.

[1]This simulator behavior may change slightly.

[2]The actual log will display all entries on a single line.