UM-Translog-2: A Planning Domain Designed for AIPS2002

Dan Wu

Department of Computer Science
University of Maryland
College Park, MD 20742
E-mail: dandan@cs.umd.edu
Jan. 8 2002

Abstract

As planning systems grow in sophistication and capabilities, planning domains with matching complexity need to be devised to assist in the analysis and evaluation of planning systems and techniques. UM Translog is a planning domain designed specially for this purpose and is a good candidate domain for AIPS planning competition.

The domain modeling language PDDL2.1 has been developed for AIPS-2002 planning competition. PDDL2.1 is intended to support representation of real time problem domains involving numeric-valued resources. UM-Translog-2 extends UM Translog by introducing some numerical computation features that make domain more realistic and suitable for used in the competition. UM-Translog-2 is encoded in PDDL2.1 for competition purpose.

1 Background and Motivation

As planning systems grow in sophistication and capabilities, planning domains with matching complexity need to be devised to assist in the analysis and evaluation of planning systems and techniques. UM Translog [1] is a planning domain designed specially for this purpose. UM-Translog provides a rich set of entities, attributes, operators and conditions, which can be used to specify rather complex planning problems with a variety of plan interactions. Due to its complexity and size, UM Translog is a good candidate domain for AIPS planning competition.

The domain modeling language PDDL2.1 [2] has been developed for AIPS-2002 planning competition. PDDL2.1 is a significant extension of PDDL intended to support representation of real time problem domains involving numeric-valued resources. Domains involving numerical computation will be more like real time problems.

UM-Translog-2 extends UM-Translog by introducing some numerical computation features that make the domain more realistic and suitable to be used in the competition. UM-Translog-2 domain is encoded in both PDDL2.1 and SHOP2 [3] domain definition syntax.

Since UM Translog was originally defined as an HTN domain and therefore had no STRIPS-style domain description. Encoding UM-Translog-2 in PDDL involves translation of HTN methods to STRIPS-style operators. In general, HTN planning is strictly more expressive than STRIPS-style planning. This means that every STRIPS-style planning problem can be translated into an HTN planning problem, but not vice versa. The kinds of HTN problems that are not translatable are those that include unbounded recursions among their methods. However, in UM Translog, there are no recursive methods at all, so it can be translated in principle into a STRIPS representation. Although possible, the whole translation process is still very complicated.

This documentation of UM-Translog-2 is based on the documentation of original version of UM-Translog domain [1].

Section 2 describes domain testing. Section 3 gives the current issues about the domain and Section 4 presents UM-Translog-2 domain in details. The domain is described in PDDL format as in action-based planner.

2. Domain Testing

UM-Translog-2 is an extension of UM Translog. However, UM Translog is in HTN format while UM-Translog-2 is in PDDL format. In order to make sure that the translation between these two formats was done correctly, we had to test the domain carefully. We take following steps to test the domain:

a. Write a random problem generator for UM-Translog-2.
b. Implement the domain for an action-based planner, TLPlan
   Actually, it would be better if we were able to choose a planner that could directly take PDDL format as input. Unfortunately, we could not find that kind of planner that could also solve the problems efficiently. Therefore, we chose to translate the domain to TLPlan format. This process is not difficult since it is an action-based to action-based translation. We also added some control formulae into TLPlan version of domain given that these formulae would not affect the correctness of the translation.
c. We implemented UM-Translog-2 domain for a HTN planner, SHOP2.
d. We ran ten problem-sets (10 problems in each set) generated by random problem generator on both TLPlan and SHOP2. For all problems we checked if both planners got the same conclusion, i.e. there exists a plan or there is no plan.
e. For those problems that both planners found the plan, we translated the problem and plan into PDDL format and used PDDL plan validator provided for the competition to check if these plans are valid.

3. Current Issues

We have devised a domain, UM-Translog-2, contributed to AIPS2002 competition, by translating UM Translog domain from HTN format to PDDL format. We also added numerical computation features into the domain so that it becomes more realistic. The domain is very complicated and is good to test the sophistication of different planning systems. However, there are still issues to be addressed, especially with regards to testing the validity of the domain:

a. Because the domain is very complicated, it is hard for the random problem generator to generate problems that are solvable with a good probability.
b. It will be better if we can find an action-based planner that takes PDDL format as input and is efficient enough to handle the domain.
c. Although we added some control formulae in TLPlan format of the domain. TLPlan is still not efficient enough to handle big problems for the domain. So we cannot test the domain in complicated cases. We can only use small problems, and try to manipulate the parameters in the random problem generator so that we can get cases that are as comprehensive as possible. We cannot add more control formulae to the domain because of the reason we mentioned in Section 2.

4 Domain Description

4.1 Overview

As in UM Translog, in UM-Translog-2, the planner is given one or more goals, where a goal is typically the delivery of a particular package from an origin to a destination.

In UM-Translog-2, we added some numerical computation features to make it more realistic and suitable for AIPS2002 competition.

In order to do this, we modeled additional aspects of transport logistics not present in the UM Translog. These include the following restrictions:

- A vehicle can be moved only with enough gas, given the newly-introduced numerical distances between locations and gpm, gasoline consumed by a vehicle per mile.
- There is no refueling for vehicles
- A vehicle cannot load packages beyond its weight and volume capacity
- A vehicle has weight, height, length and width
- A package has weight and volume.
- An equipment like crane cannot pick up a package beyond its weight and volume capacity
- A route cannot accommodate a vehicle beyond its height and weight capacity
- A location cannot accommodate packages beyond its volume capacity
- A location cannot accommodate vehicle beyond its length, height and width capacity

Domain is described in more details in the following sections. Section 4.2 introduces entities. Predicates and functions are described in section 4.3 and operators are described in section 4.4.

4.2 Entities

Entities include regions, cities within each region, locations within each city and individual objects (routes, vehicles, equipment, and packages). Each entity is described by a constant symbol (e.g., "Truck-1", "Package-2") and one or more functions and predicates that are asserted by a user (in the initial state given to the planner) or by the effects of instantiated plan operators. Predicates and functions are summarized in section 4.3. Each entity has a type. Primary entity types include region, city, location, route, vehicle, equipment and package, described in the following subsections.

4.2.1 Region

Each region contains one or more cities (specified via predicate **in-region**).

4.2.2 City

Each city can have one or more locations (specified via predicate **in-city**).

4.2.3 Location

Each location is located in a specific city (specified via predicate **in-city**).

Location subtypes include transportation centers (specified via predicate **tcenter**) and non-transportation centers. Transportation center subtypes (specified via predicate **typel**) include **airport** and **train-station**. Non-transportation centers denote customer locations, such as businesses, homes, etc.

A transport center can be used for air/rail direct and indirect transportation (see section 4.4). Transportation centers can be available (specified via predicate **availablel**) or unavailable. For example, a particular airport may be temporarily unavailable due to bad weather.

A transportation center can optionally be specified as a transportation hub (via **hub** predicate). Hub transportation centers can be used for indirect transportation (see section 4.4). A transportation center serves its own city. Thus, air or rail travel from a specific city must use a transportation center in that city. Hub transport centers serve specific regions (specified via predicate **serves**), rather than cities. A hub serves a region if it has rail/air route connection to a transport center in that region.

Locations can serve as the origin or destination of a package. Locations have their volume capacity (specified via function **volume-cap-l**). The total volume of all packages (see section 4.2.7) in a location (specified via function **volume-load-l**) at any given time cannot exceed its volume capacity.

Also, a location cannot accommodate a vehicle (see Section 4.2.5) whose length exceeds location's length capacity (specified via function **length-cap-l)** or whose width exceeds location's width capacity (specified via function **width-cap-l)** or whose height exceeds location's height capacity (specified via function **height-cap-l**). The distance between any two locations is specified via function **distance**.

4.2.4 Routes

Route includes types **road-route**, **rail-route**, and **air-route**.

Road routes connect two cities (specified via predicate **connect-city)**. All locations within a city are assumed to be connected by roads, and thus road routes between individual city locations are not specified. Rail and air routes connect airports and train stations, respectively (specified via predicate **connect-loc)**.

Routes have an origin, a destination, and a route type (specified via predicate **connect-city** or **connect-loc**). Note that routes are directional: traffic flows from the origin to the destination. Route has an availability status (specified via predicate **availabler**). For example, a particular road route may be temporarily unavailable due to construction. Routes types are compatible with particular types of vehicles (see Section 4.2.5), as follows:

Route Type	Vehicle Type
road-route	**truck**
rail-route	**train**
air-route	**airplane**

Route-vehicle type compatibilities are specified via predicate **rv-compatible**.

A route cannot be used by a vehicle whose height exceeds route's height capacity (specified via function **height-cap-r**) or whose total weight (including vehicle weight and load) exceeds route's weight capacity (specified via function **weight-cap-r**). The height and weight capacity of local roads within a city are specified via functions **local-height** and **local-weight**.

4.2.5 Vehicle Types

Primary vehicle types include **truck**, **airplane** and **train** (specified via predicate **typevp**). Each vehicle also has a physical subtype (specified via predicate **typev**). The physical subtype for airplane is **air**, and the physical subtype for trucks and trains are as following:

Physical Subtype	Examples
regularv	tractor-trailer truck, delivery van, boxcar, etc.
flatbed	flatbed truck, flatcar, etc.
tanker	tanker truck, tanker car, etc.
hopper	dump truck, hopper car, etc.
auto	car carrier truck/train

A vehicle's primary type determines its compatibility with a particular route (see Section 4.2.4), while its physical subtype determines its compatibility with a package (see Section 4.2.7).

A vehicle is at a location and has availability status (specified via predicates **at-vehicle** and **availablev**, respectively). A vehicle may have other properties, depending on its subtype, as shown in the following table:

Physical Subtype	Predicates
air	**door-open, ramp-connected**

auto	**ramp-down**
hopper	**chute-connected**
regularv	**door-open**
tanker	**hose-connected, valve-open**

A vehicle has weight (specified via function **weight-v**), length (specified via function **length-v**), width (specified via function **width-v**) and height (specified via function **height-v**).

A vehicle consumes gas when moving. The gas-consumption rate of a vehicle is specified via function **gpm** (gallon per mile). A vehicle can be moved between two locations only if we have:

Gas left in the vehicle (specified via function **gas-left**) >= vehicle's **gpm** * distance between two locations.

The total volume of all packages in a vehicle (specified via function **volume-load-v**) cannot exceed its volume capacity (specified via function **volume-cap-v**) and the total weight of all packages in a vehicle (specified via function **weight-load-v**) cannot exceed its weight capacity (specified via function **weight-cap-v**).

4.2.6 Equipment Types

Equipment types are **plane-ramp** and **crane**. Equipments of these types are used to load airplanes and flatbed trucks/trains, respectively.

An equipment is at a location (specified via predicate **at-equipment**). And there is no action that changes the location of an equipment.

The status of a plane ramp is described using predicate **ramp-connected**.

The status of a crane is described using predicate **empty**. Also a crane cannot pick up a package beyond its weight capacity (specified via function **weight-cap-c**) or volume capacity (specified via function **volume-cap-c**).

4.2.7 Package Types

Each packages has a physical subtype from the following list (specified via predicate **typep**)

Physical Subtype	Examples
regularp	parcels, furniture, etc.
bulky	steel, lumber, etc.
liquid	water, petroleum, chemicals, etc.
granular	sand, ore, etc.
cars	automobiles
mail	mail

The physical subtype of a package must be compatible with the vehicle's physical subtype (see Section 4.2.5). The following table lists compatible package and vehicle physical subtypes (specified via predicate **pv-compatible**):

Package Subtype	Vehicle Subtype
regularp	**regularv**
bulky	**flatbed**
liquid	**tanker**
granular	**hopper**
cars	**auto**
regularp	**air**
mail	**air, regularv**

Each package has a location (specified via predicate **at-package**), weight (specified via function **weight-p**) and volume (specified via function **volume-p**). Fees need to be collected before a package can be transported (specified via predicate **fees-collected)**

When package is at its destination, it will be delivered (specified via predicate **delivered**).

4.3 Predicates and Functions

This section presents a summary of domain predicates and functions present in the PDDL version.

The following are the domain predicates:

Predicates	Descriptions
(at-equipment ?e - equipment ?l - location)	equipment ?e is at location ?l
(at-packagec ?p - package ?c - crane)	package ?p is at crane ?c
(at-packagel ?p - package ?l - location)	package ?p is at location ?l
(at-packagev ?p - package ?v - vehicle)	package ?p is at vehicle ?v
(at-vehicle ?v - vehicle ?l - location)	vehicle ?v is at location ?l
(availablel ?l - location)	location ?l (a transport center) is available
(availabler ?r - route)	route ?r is available
(availablev ?v - vehicle)	vehicle ?v is available
(chute-connected ?v - vehicle)	chute of vehicle ?v (hopper) is connected to (un)load cargo
(clear)	bookkeeping predicate in the domain (see section 4.4)
(connect-city ?r - route ?rtype - rtype ?c1 ?c2 - city)	route ?r of type ?rtype connects city ?c1 to city ?c2
(connect-loc ?r - route ?rtype - rtype ?l1 ?l2 - location)	route ?r of type ?rtype connects location ?l1 to location ?l2
(delivered ?p - package ?d - location)	package ?p is delivered at location ?d
(door-open ?v - vehicle)	door of vehicle ?v is open
(empty ?c - crane)	crane ?c is empty
(fees-collected ?p - package)	fees have been collected for package ?p
(hose-connected ?v - vehicle)	hose connected for ?v (tanker) to (un)load cargo
(h-start ?p - package)	bookkeeping predicate in the domain (see section 4.4)
(hub ?l - location)	location ?l is a hub
(in-city ?l - location ?c - city)	location ?l is located in city ?c
(in-region ?c - city ?r - region)	city ?c is inside region ?r
(move ?p - package) / (move-emp ?v - vehicle) / (over ?p - package)	bookkeeping predicate in the domain (see section 4.4)
(pv-compatible ?ptype - ptype ?vtype - vtype)	package physical subtype ?ptype is compatible with vehicle physical subtype ?vtype
(ramp-connected ?v - vehicle ?r - plane-ramp)	plane ramp ?r is connected to vehicle ?v (airplane)
(ramp-down ?v - vehicle)	ramp of vehicle ?v (auto) is down to (un)load cargo
(rv-compatible ?rtype - rtype ?vptype - vptype)	route type ?rtype is compatible with primary vehicle type ?vptype
(serves ?h - location ?r - region)	location ?l (hub) serves region ?r

(tcenter ?l - location)	location ?l is tcenter
(t-end ?p - package) / (t-start ?p - package)	bookkeeping predicate in the domain (see section 4.4)
(typel ?l - location ?type - ltype)	location ?l (tcenter) is of type ?type (train station or airport)
(typep ?p - package ?type - ptype)	package ?p has physical subtype ?type
(typev ?v - vehicle ?type - vtype)	vehicle ?v has physical subtype ?type
(typevp ?v - vehicle ?type - vptype)	vehicle ?v has primary type ?type (truck, train, airplane)
(unload ?v - vehicle)	bookkeeping predicate in the domain (see section 4.4)
(valve-open ?v - vehicle)	valve open for vehicle ?v (tanker) to (un)load cargo

The following are the domain functions:

Functions	Descriptions
(distance ?l1 ?l2 - location)	distance between two locations ?l1 and ?l2
(gas-left ?v - vehicle)	gallons of gas left in vehicle ?v
(gpm ?v - vehicle)	gallons of gas ?v consumes per mile
(height-v ?v - vehicle)	height of vehicle ?v in feet
(height-cap-l ?l - location)	height capacity of location ?l in feet
(height-cap-r ?r - route)	height capacity of route ?r in feet
(length-v ?v - vehicle)	length of vehicle ?v in feet
(length-cap-l ?l - location)	length capacity of location ?l in feet
(local-height ?c - city)	height capacity of local road route in city ?c in feet
(local-weight ?c - city)	weight capacity of local road route in city ?c in pounds
(volume-cap-c ?c - crane)	volume capacity of crane ?c in liters
(volume-cap-l ?l - location)	volume capacity of location ?l in liters
(volume-cap-v ?v - vehicle)	volume capacity of vehicle ?v in liters
(volume-load-l ?l - location)	total volume of packages at location ?l in liters
(volume-load-v ?v - vehicle)	total volume of packages in vehicle ?v in liters
(volume-p ?p - pakcage)	volume of package ?p in liters
(weight-cap-c ?c - crane)	weight capacity of crane ?c in pounds
(weight-cap-r ?r - route)	weight capacity of route ?r in pounds
(weight-cap-v ?v - vehicle)	weight capacity of vehicle ?v in pounds
(weight-p ?p - package)	weight of package ?p in pounds
(weight-load-v ?v - vehicle)	total weight of packages in vehicle ?v in pounds
(weight-v ?v - vehicle)	weight of vehicle ?v in pounds
(width-v ?v - vehicle)	width of vehicle ?v in feet
(width-cap-l ?l - location)	width capacity of location ?l in feet

4.4 Operators

This section describes the symbols that denote operators in UM-Translog-2. Although UM-Translog-2 is based on UM Translog, the operators in these two domains are quite different. UM translog is developed for HTN planning systems while UM-Translog-2 is written in action-based format for competition purpose. Some bookkeeping predicates are needed during the translation process as described below.

4.4.1 Administrative Operators

Prior to carrying a package to its destination, fees should be collected. Each package must be delivered to its destination. These activities are denoted by the operator symbols **collect-fees(?p)** and **deliver(?p, ?l)**, where ?p is a variable symbol denoting a package and ?l is a variable symbol denoting a location. Fees for a package need to be collected only once, and a package can be delivered only once.

4.4.2 Operators for Loading/Unloading

There are a number of operators for loading and unloading packages into/from vehicles, depending on the type of the vehicle and the package. In some cases, special equipment such as crane needs to be used for that purpose.

Before loading a regular vehicle, the door of the vehicle must be open and after loading all packages, the door of the vehicle must be closed. These steps are denoted by actions **open-door-regular(?v)**, **load-regular(?p ?v ?l)**, **close-door-regular(?v)**. Unloading a regular vehicle involves the same steps, just replacing **load-regular(?p, ?v, ,?l)** with **unload-regular(?p ?v ?l)**. ?p is a variable of type package, v? is a variable of type vehicle, and ?l is a variable of type location. ?l is used to make sure the vehicle and the package are at the same location.

Loading a flatbed requires sequence of actions **pick-up-package-ground(?p ?c ?l)** and **put-down-package-vehicle(?p ?c ?v ?l)**. Unloading a flatbed requires sequence of actions **pick-up-package-vehicle(?p ?c ?v ?l)** and **put-down-package-ground(?p ?c ?l)**. ?c denotes crane needed for loading and unloading the flatbed.

Before loading a truck or train of type hopper, the chute of the vehicle must be connected and after loading all packages, the chute must be disconnected. These steps are denoted by actions **connect-chute(?v)**, **fill-hopper(?p ?v ?l)**, and **disconnect-chute(?v)**. Unload is similar, except that **empty-hopper(?p ?v ?l)** should be replaced with **fill-hopper(?p ?v ?l)**.

Before loading a vehicle of type tanker, the hose of the vehicle must be connected first and then the valve of the vehicle needs to be open. After loading all packages, the valve must be closed first and then the host must be disconnected. These steps are denoted by actions **connect-hose(?v)**, **open-valve(?v)**, **fill-tank(?v ?p ?l)**, **close-valve(?v)**, **disconnect-hose(?v ?p)**. Unload is similar, except that **fill-tank(?v ?p ?l)** should be replaced with **empty-tank(?v ?p ?l)**.

Before loading a vehicle of type auto, the ramp of the vehicle must be lowered and after loading all packages, the ramp must be raised. These steps are denoted by actions **lower-ramp(?v)**, **load-**

cars(?p ?v ?l) and **raise-ramp(?v)**. Unloading is similar, except that **load-cars(?p ?v ?l)** should be replaced with **unload-cars(?p ?v ?l)**.

Before loading a vehicle of type air, a conveyor ramp must be attached to the vehicle first and then the door of the vehicle must be open. After loading vehicles, the door must be closed first and then the ramp needs to be detached. These steps are denoted by actions **attach-conveyor-ramp(?v, ?r, ?l)**, **open-door-airplane(?v)**, **load-airplane(?p, ?v, ?l)**, **detach-conveyor-ramp(?v, ?r, ?l)** and **close-door-airplane(?v)**. Unloading is similar, except that **load-airplane(?p, ?v, ?l)** should be replaced with **unload-airplane(?p, ?v, ?l)**.

In the effect list of operators for unloading a vehicle, there are some special predicates used for bookkeeping purpose as explained below:

a. **(not (move ?p))**
   As a rule in UM Translog domain (see section 4.4.3 for more explanation), each movement of a package ?p from a location ?l1 to a location ?l2 by using a vehicle ?v involves three steps: loading ?p into ?v at ?l1, moving ?v from ?l1 to ?l2 and unloading ?p from ?v at ?l2. This means that ?p must be unloaded at ?l2 before it can be moved further more. So after each movement of ?v from ?l1 to ?l2, predicate **(move ?p)** will be added to the current state, and after ?p is unloaded at ?l2, this predicate will be removed from current state which means ?p can be moved again.
b. **(unload ?v)** and **(not (clear))**
   After our task is finished, we need to make sure that all things are cleaned up after us. For example, we should close the door of all regular vehicles we have used, raise the ramps of all auto vehicles we have used, etc. **(clear)** is a predicate used to indicate that all things have been cleaned up after us. **(unload ?v)** means that we have used vehicle ?v and need to do some clean up stuff for ?v. So in the effect of unloading operators, **(unload ?v)** is added to the current state and **(clear)** is deleted from the current state. **(clear)** can be added to the current state by **clean-domain** operator(see section 4.4.4) when there is nothing which needs to be cleaned up. **(clear)** is the goal of each problem of the domain.

4.4.3 Operators for Moving

In UM Translog domain, there are some rules about how to move a package from its origin to its destination. This involves choosing a suitable path (a sequence of routes from the origin to the destination), and moving the package along that path via a series of carry-direct tasks.

A (carry-direct ?package ?location1 ?location2) task involves picking a route directly connecting ?location1 and ?location2, and choosing a vehicle that is compatible both with the package and the route. Only those vehicles that are at ?location1 or one step away from ?location1 (which means that this vehicle can be moved from its location to location1 directly without passing by any other locations) can be used. The task is accomplished by moving that vehicle to ?location1, loading the package into the vehicle, moving the vehicle to ?location2, and finally unloading the package. When a vehicle moves, so do the packages it contains.

The diagram in Figure 1 shows the legal paths to transport a package. The origin of the package can be either clocation1 (a customer location, not a transportation center) or tcenter1 (a transportation center), and similarly the destination of a package can be either clocation2 (a

customer location, not transportation center) or tcenter2 (a transportation center). There are some additional rules about this path:

1. clocation1 can only use a transportation center (tcenter1) in the same city, so does clocation2
2. tcenter1 and tcenter2 can not be hubs if hub1 is used.
3. The route that connects tcenter1 and hub1 is a rail/air route.
4. The route that connects hub1 and tcenter2 is a rail/air route.
5. If a package is transported from clocation1 or transported to clocation2 using a route between tcenter1 and tcenter2, then this route must be a rail/air route.

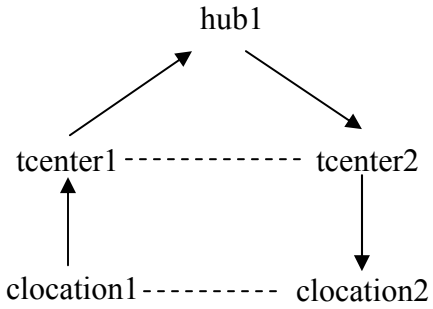

Figure 1 Transport Path

All possible legal pathes for transporting a package are defined more precisely as follows.

A package p must be transported from origin ?ori to destination ?des through one of following pathes:

a. If ?ori and ?des are in the same city c, use local road route in city c.
b. If ?ori and ?des are in two different cities c1, c2, use a road route r that connects c1 and c2.
c. If ?ori and ?des are both train stations, use a rail route r that connects ?ori and ?des.
d. If ?ori and ?des are both airports, use an air route r that connects ?ori and ?des.
e. If ?ori and ?des are both tcenters (train station or airport), but are both not hub and hub hub1 is of same type as ?ori and ?des (train station or airport), then
   transport p from ?ori to hub1 use method c or d
   transport p from hub1 to ?des use method c or d
f. If ?ori is not tcenter, ?des is tcenter, and
   ?ori is in city c1 and
   tcenter1 is a transportation center in c1 and tcenter1 is of same type as ?des, then
   transport p from ?ori to tcenter1 use method a
   transport p from tcenter1 to ?des use method c, d or e
g. If ?ori is tcenter, ?des is not tcenter, and
   ?des is city c2, and
   tcenter2 is a transportation center in c2 and tcenter1 is of same type as ?ori, then
   transport p from ?ori to tcenter2 use method c, d or e
   transport p from tcenter2 to ?des use method a
h. If ?ori is not tcenter, ?des is not tcenter, and
   ?ori is in city c1 and and ?des is in city c2 (c1 and c2 can be the same city), and
   tcenter1 is a transportation center in c1, tcenter2 is a transportation center in c2 and
   tcenter1, tcenter2 are of same type, then
   transport p from ?ori to tcenter1 use method a

transport p from tcenter1 to tcenter2 use method c, d or e
transport p from tcenter2 to ?des use method a

In UM-Translog-2 domain, we still follow the rules as described above. In order to make sure that a package is transported along a legal path, we have to keep track of the movement of a package in an action-based planner. Following predicates are used for this bookkeeping purpose. Variable ?p is of type package.

Predicates	Meaning
(over ?p)	?p cannot be moved anymore according to Figure 1
(t-start ?p)	?p is at tcenter1 according to Figure1
(t-end ?p)	?p is at tcenter2 according to Figure1
(h-start ?p)	?p has visited one hub and is at hub1 or tcenter2 (when tcenter2 is hub and hub1 is not used in the path) as shown in Figure1.

In order to keep track of the movement of a package, we also divided the movement of a vehicle into different cases and have following vehicle moving operators (variable ?v denotes vehicle, variable ?ori denotes the origin, variable ?des denotes the destination):

1. When moving ?v using local-road-route within a city ?ocity, we have following operators:
    a. **move-vehicle-local-road-route1 (?v, ?ori, ?des, ?ocity)** for the case that either ?ori and ?des are both transportation centers or are both non-transportation centers
    Before using this operator, none of the packages inside the vehicle have been moved ever and after using this operator, none of the packages inside the vehicle can be moved anymore (i.e. predicate (over ?p) is added in the current state for all packages inside the vehicle).
    b. **move-vehicle-local-road-route2 (?v, ?ori, ?des, ?ocity)** for the case that ?ori is not a transportation center and ?des is one
    Before using this operator, none of the packages inside the vehicle have been moved ever, and after using this operator, all packages inside the vehicle are at point tcenter1 in Figure1 (i.e. predicate (t-start ?p) is added in the current state for all packages inside the vehicle).
    c. **move-vehicle-local-road-route3 (?v, ?ori, ?des, ?ocity)** for the case that ?ori is a transportation center and ?des is not one
    According to Figure1, before using this operator, either none of the packages inside the vehicle have been moved ever, or all of them must be at tcenter2 (with predicate (h-start ?p) or (t-end ?p)) and after using this operator, none of packages inside the vehicle can be moved anymore.
2. When moving ?v using road-route ?r which connects two different cities ?ocity and ?dcity, we have operator
    move-vehicle-road-route-crossCity(?v, ?ori, ?des, ?ocity, ?dcity, ?r)
    Before using this operator, none of the packages inside the vehicle have been moved ever and after using this operator, none of the packages inside the vehicle can be moved anymore.
3. When moving ?v using a rail or air route ?r, we have following operators

a. **move-vehicle-nonroad-route1(?v, ?ori, ?des, ?r)** for the case that either ?ori and ?des are both hubs or are both not hubs
   Before using this operator, either none of the packages inside the vehicle have been moved ever or all of them must be at tcenter1 in Figure 1 and after using this operator, all packages inside the vehicle are at tcenter2 in Figure1.
b. **move-vehicle-nonroad-route2(?v, ?ori, ?des, ?r)** for the case that ?ori is not a hub and ?des is a hub
   According to Figure1, before using this operator, either none of the packages inside the vehicle have been moved ever or all of them must be at tcenter1 (with predicate (t-start ?p)) and after using this operator, all packages inside the vehicle are at either hub1 or tcenter2 (with predicate (h-start ?p)).
c. **move-vehicle-nonroad-route3(?v, ?ori, ?des, ?ocity)** for the case that ?ori is a hub and ?des is not a hub
   According to Figure1, before using this operator, either none of the packages inside the vehicle have been moved ever or all of them must be at tcenter1 or hub1 (with predicate (t-start ?p) or (h-start ?p)) and after using this operator, all packages inside the vehicle are at tcenter2 (with predicate (t-end ?p)).

In both preconditions and effects of all moving operators, we have a predicate (move-emp ?v) where ?v is a variable symbol denoting a vehicle. The reason for using this predicate is that in UM Translog, there is a rule saying that if a package needs to be moved from a location and there is no vehicle at this location, then only those vehicles that are one step away from the current location can be used to move this package. What this rule means is that if an empty vehicle is moved to a location, it cannot be moved anymore before it picks up something from this location. This is guaranteed through:

a. If we use moving operators to move an empty vehicle ?v, (move-emp ?v) predicate will be added to the current state.
b. In moving operators, (not (move-emp ?v)) is used as a precondition for empty vehicle.
c. (move-emp ?v) will be deleted from the current state after ?v is moved as a non-empty vehicle.

4.4.4 Clean Domain

We also have an operator **clean-domain()**. This operator is used to check if we have cleaned up after us, that is, if we have closed doors of all regular vehicles we have used, disconnected chutes of all tankers we have used, etc. This operator is applicable if everything is cleaned up and predicate (clear) will be added to the current state. (clear) is also the goal of every problem in the domain.

Reference

[1] S. Andrews, B. Kettler, K. Erol and J. Hendler. "UM Translog: A Planning Domain for the Development and Benchmarking of Planning Systems." Tech. Report CS-TR-3487, Dept. of Computer Science, University of Maryland, College Park, MD, 1995.

[2] Maria Fox and Derek Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains." Tech. Report, University of Durham, UK, 2001.

[3] D. Nau, H. Muñoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. "Total-Order Planning with Partially Ordered Subtasks." In *IJCAI-2001*. Seattle, August, 2001.