UM Translog: A Planning Domain for the Development and Benchmarking of Planning Systems [*]

Scott Andrews, Brian Kettler, Kutluhan Erol, and
James Hendler

Department of Computer Science,
Institute for Advanced Computer Studies,
and Institute for Systems Research
University of Maryland
College Park, MD 20742
E-mail: hendler@cs.umd.edu
June 1995

---

[*] This research was supported in part by grants from NSF (IRI-8907890), ONR (N00014-J-91-1451), AFOSR (F49620-93-1-0065), the ARPA/Rome Laboratory Planning Initiative (F30602-93-C-0039) and by ARI (MDA-903-92-R-0035), subcontract through Microelectronics and Design, Inc. Dr. Hendler is also affiliated with the Institute for Systems Research and the Institute for Advanced Computer Studies.

Abstract

The last twenty years of AI planning research has discovered a wide variety of planning techniques such as state-space search, hierarchical planning, case-based planning and reactive planning. These techniques have been implemented in numerous planning systems (e.g., [12, 8, 9, 10, 11]). Initially, a number of simple toy domains have been devised to assist in the analysis and evaluation of planning systems and techniques. The most well known examples are "Blocks World" and "Towers of Hanoi." As planning systems grow in sophistication and capabilities, however, there is a clear need for planning benchmarks with matching complexity to evaluate those new features and capabilities. UM Translog is a planning domain designed specifically for this purpose.

UM Translog was inspired by the CMU Transport Logistics domain developed by Manuela Veloso. UM Translog is an order of magnitude larger in size (41 actions versus 6), number of features and types interactions. It provides a rich set of entities, attributes, actions and conditions, which can be used to specify rather complex planning problems with a variety of plan interactions. The detailed set of operators provides long plans (4̃0 steps) with many possible solutions to the same problem, and thus this domain can also be used to evaluate the solution quality of planning systems. The UM Translog domain has been used with the UMCP, UM Nonlin, and CaPER planning systems thus far.

Contents

1 Background and Motivation **3**
1.1 A Guide to This Document 3

2 Overview **5**

3 Entities **6**
3.1 Location Types 6
3.2 Routes Types 7
3.3 Vehicle Types 7
3.4 Equipment Types 8
3.5 Package Types 8

4 Predicates **11**

5 Actions **13**
5.1 Administrative Actions 13
5.2 Actions for loading/unloading 13
5.3 Transit Actions 14

6 Tasks **15**

A UM Nonlin Domain Definition **18**
A.1 Nonlin-specific Domain Predicates 18
A.2 Specifying the Problem 19
A.2.1 Locations 20
A.2.2 Routes 20
A.2.3 Vehicles 20
A.2.4 Equipment 21
A.2.5 Packages 21
A.3 A Sample Problem 22
A.3.1 Always Context Forms: 22
A.3.2 Initial Context Forms: 23
A.3.3 Goal Forms: 24
A.3.4 Plan Output by Nonlin: 24

B UMCP Domain Definition **25**
B.1 Problem Specification 25

1 Background and Motivation

The last twenty years of AI planning research has discovered a wide variety of planning techniques such as state-space search, hierarchical planning, case-based planning and reactive planning. These techniques have been implemented in numerous planning systems (e.g., [12, 8, 9, 10, 11]). Initially, a number of simple toy domains have been devised to assist in the analysis and evaluation of planning systems and techniques. The most well known examples are "Blocks World" and "Towers of Hanoi." As planning systems grow in sophistication and capabilities, however, there is a clear need for planning benchmarks with matching complexity to evaluate those new features and capabilities. UM Translog is a planning domain designed specifically for this purpose.

UM Translog was inspired by the CMU Transport Logistics domain developed by Manuela Veloso[7]. UM Translog is an order of magnitude larger in size (41 actions versus 6), number of features and types interactions. It provides a rich set of entities, attributes, actions and conditions, which can be used to specify rather complex planning problems with a variety of plan interactions. The detailed set of operators provides long plans ($\tilde{4}0$ steps) with many possible solutions to the same problem, and thus this domain can also be used to evaluate the solution quality of planning systems.

UM Translog is currently being used in the evaluation of a case-based planning system, CaPER [6, 5], and a hierarchical task network planning system, UMCP [3, 2]. It is also being used by UM Nonlin [4], a common lisp implementation of Austin Tate's Nonlin HTN planning system, to generate a plan library for plan reuse by CaPER.

Due to the complexity and size of the domain, it is not easy to define UM Translog problems manually. In order to facilitate the use of UM Translog, a system has been implemented for generating random problems or assisting the user in defining his/her own problems. This system can be adapted to the input format requirements of other planning systems with reasonable effort. We hope to make the Problem Specifier/Generator code available soon.

Documentation and code related to UM Translog, UM Nonlin, UMCP and CaPER can be accessed through Worldwide Web (WWW) at URL http://www.cs.umd.edu/projects/plus/[1] or by anonymous FTP.[2]

Because several of these planning systems are ongoing research projects, the domain definition may change. We are distributing this domain definition and the affiliated code for using it with UM Nonlin and UMCP free of charge but without any implied guarantees or promises of support.

We hope that other researchers will contribute their planning domains so that a library of benchmark planning domains can be established, similar to the benchmark library used by the machine learning community.

1.1 A Guide to This Document

Section 2 presents briefly the our goals in defining the UM Translog domain and some of its general features. The domain is then described in a planner-independent manner in the sections that follow.

---

[1] The UM Translog homepage is at URL http://www.cs.umd.edu/projects/plus/UMT/.

[2] Instructions for FTPing UM Translog can be obtained by anonymous FTP to ftp.cs.umd.edu (file is /pub/nonlin/um-translog/README).

Entities and predicates are described in Sections 3 and 4, respectively. Operators are described in Section 5 (Actions) and Section 6 (Tasks/Methods). Some planner-specific information about this domain as defined for the UM Nonlin and UMCP systems is given in the appendices (details can be found in the actual domain definitions available from the WWW site or via FTP).

2 Overview

In this domain, 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. Our goal for UM Translog domain was to create a domain more complex than toy domains such as blocks world. To do this we modelled additional aspects of transport logistics not in the CMU Transport Logistics domain, and which we believed were somewhat realistic. These include the following features and restrictions:

- transport is by air, rail, or road
- transport is intracity or intercity
- several basic methods of transport are available including local transport via road, direct transport via a specific direct route, and "indirect" transport via a transportation hub
- customer locations and transportation centers (airports and train stations) are grouped into cities which are grouped into regions
- intercity transport uses specific routes
- transport via air or rail uses specific transportation centers (airports and train stations)
- particular vehicles and packages have special (un)loading methods and actions
- packages and vehicles have (sub)types which must be compatible
- vehicles, equipment, routes, and transportation centers may be temporarily unavailable
- certain cities may not allow hazardous packages to be transported through them

Our goal, however, was *not* to model "real" transport planning as described by a domain expert, for example. Obviously the kind of transport logistic planning done in the "real world" by UPS, the military, etc. is considerably more complicated, particularly as the world state is highly dynamic and often not fully known by the planner. Some major aspects of transport logistics planning we are *not* modelling in order to keep the domain "manageable" and to stay with in the capabilities of the planning systems we are testing include:

- package sizes and vehicle capacities (We assume unlimited vehicle capacity.)
- numeric distances between locations[3]
- temporal aspects including delivery deadlines, action durations (travel time, etc.), action concurrency, etc.

---

[3] Distances are "simulated" to some extent by the grouping of locations into cities and cities into regions and the explicit specification of intercity routes.

3 Entities

UM Translog entities include individual locations (cities, etc.) 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 predicates that are asserted by a user (in the initial state given to the planner) or by the effects of instantiated plan actions.[4] Predicates are summarized in Section 4. Each entity has a primary type, specified by the predicate **type**. Primary entity types include location types, route types, vehicle types, equipment types, and package types, described in the following subsections.

3.1 Location Types

As shown in Figure 1, location types are **region**, **city**, and **city-location**. City location subtypes are **tcenter** (transport center) and **not-tcenter** (a city location that is *not* a transport center). Transport center subtypes are **airport** and **train-station**. Non-transport center subtypes are **clocation** (customer location) and **post-office**.

Figure 1: Location Type Hierarchy

Region contain one or more cities (specified via the predicate **in-region**).

Cities contain one or more city locations (specified via the predicate **in-city**). Some cities are compatible with hazardous packages (specified via the predicate **pc-compatible**, see Section 3.5).

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

A transport center can be used for air/rail direct and indirect transport (see Section 6). Transport centers can optionally be specified as transport hubs (via the **hub** predicate). Hub transport centers can be used for indirect transport (see Section 6). Transport centers serve specific cities (specified via the predicate **serves**). Thus air or rail travel from a specific city must use a transport center that serves that city. Hub transport centers serve specific *regions*, rather than cities. Trans-

---

[4] Domain constants and predicates are shown in boldface in this document.

port centers can be available or not (specified via the subtype **hub**). For example, a particular airport may be temporarily unavailable due to bad weather.

Customer locations are "generic" locations (e.g., businesses, homes, etc.) within a city that can serve as the origin and destination of a package, as can transport centers. A customer location is located in a specific city (specified via the predicate **in-city**).

A post office is similar to a customer location but can be used as the origin and destination for packages of type **mail**.

3.2 Routes Types

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

Routes connect locations. Road routes connect two cities. All locations within a city are assumed to be connected by roads, and thus road routes are are not specified between individual city locations. Rail and air routes connect airports and train stations, respectively. Routes have an origin, destination, and route type (specified via the predicate **connects**). Note that routes are directional: traffic flows from the origin to the destination. Routes have an availability status (specified via the predicate **available**). For example, an particular road route may be temporarily unavailable due to construction.

Routes types are compatible with particular types of vehicles (see Section 3.3) as follows:

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

Route–vehicle type compatibilities are specified to the planner via the predicate **rv-compatible**.

3.3 Vehicle Types

Primary vehicle types are **truck**, **airplane**, **train** (train engine), and **traincar**.

Trucks and traincars have subtypes: a single physical subtype and an optional specialty subtype. Physical truck/traincar subtypes are:

Physical Subtype	*Examples*
regular	tractor-trailer truck, delivery van, boxcar, etc.
flatbed	flatbed truck, flatcar, etc.
tanker	tanker truck, tanker car, etc.
hopper	dump truck, hopper car, etc.
mail	mail truck, mail car, etc.
livestock	livestock truck, cattle car, etc.
auto	car carrier truck/traincar

Specialty truck/traincar subtypes are:

Specialty Subtype	*Examples*
refrigerated	refrigerated truck/traincar, etc.
armored	armored truck/traincar, etc.

Specialty subtypes cannot be specified if the truck or traincar has a physical type of **mail**, **livestock** or **auto**. The physical subtype of a truck, traincar, or airplane must be compatible with the physical subtype of a package (see Section 3.5). When specified, a package specialty subtype and truck/traincar specialty subtype, must also be compatible.

Vehicles of type **train** — train engines – unlike other types of vehicles, do not hold packages themselves but rather have attached traincars that hold packages.

A vehicle's primary type determines its compatibility with particular routes (see Section 3.2). Vehicles have a location and availability (specified via the predicates **at-vehicle** and **available**, respectively).

A vehicle may have other properties, depending on its type and subtype, as shown in the following table:

Vehicle (Sub)type	*Predicates*
airplane	**door-open**, **ramp-connected**
armored	**guard-inside**, **guard-outside**
auto	**ramp-down**
hopper	**chute-connected**
livestock	**clean-interior**, **door-open**, **ramp-down**, **trough-full**
regular	**door-open**
tanker	**hose-connected**, **valve-open**
hazardous pkg. carrier	**decontaminated-interior**, **warning-signs-affixed**

3.4 Equipment Types

Equipment types are **plane-ramp** and **crane**. Equipment of these types is used to load planes and flatbed trucks/traincars, respectively.

Equipment has a location (specified via the predicate **at-equipment**).

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

The status of a crane is additionally described using the predicate **empty**.

3.5 Package Types

Packages have type **Package**. Packages have subtypes: a single physical subtype and, optionally, one or more specialty subtypes. Physical package subtypes are:

Physical Subtype	*Examples*
regular	parcels, furniture, etc.
bulky	steel, lumber, etc.
liquid	water, petroleum, chemicals, etc.
granular	sand, ore, etc.
mail	letters sent through the postal service
livestock	cattle, etc.
auto	automobiles

Specialty package subtypes are:

Specialty Subtype	*Examples*
perishable	produce, frozen food, etc.
hazardous	petroleum, nuclear waste, etc.
valuable	money, weapons, etc.

Specialty subtypes cannot be specified if the package has a physical subtype of **mail**, **livestock**, or **auto**.

The following table shows some plausible combination of package subtypes for some kinds of packages:

Kind of Package	*Set of Package Subtypes*
mail	**mail**
automobiles	**auto**
livestock	**livestock**
parcels	**regular**
goods	**regular**
furniture	**regular**
appliances	**regular**
dry food	**regular**
newspapers	**regular**
steel	**bulky**
lumber	**bulky**
machinery	**bulky**
aircraft	**bulky**, **valuable**
nuclear weapon	**bulky**, **valuable**, **hazardous**
grain	**granular**
sand	**granular**
dirt	**granular**
gravel	**granular**
ore	**granular**
coal	**granular**
uranium ore	**granular**, **hazardous**
water	**liquid**
milk	**liquid**, **perishable**
chemicals	**liquid**, **perishable**, **hazardous**
petroleum	**liquid**, **hazardous**
produce	**regular**, **perishable**
nuclear waste	**regular**, **hazardous**
medical waste	**regular**, **hazardous**, **perishable**
money	**regular**, **valuable**
art	**regular**, **valuable**

The physical subtype of a package must be compatible with the vehicle's primary type and any physical subtype specified for that vehicle (see Section 3.3). The following table lists compatible package and vehicle types (specified to the planner via the predicate **pv-compatible**):

Package Physical Subtype	*Vehicle (Sub)type*
regular	**regular**
bulky	**flatbed**
liquid	**tanker**
granular	**hopper**
perishable	**refrigerated**
valuable	**armored**
mail	**mail**
livestock	**livestock**
auto	**auto**
regular	**airplane**
mail	**airplane**

Packages with specialty subtype of **hazardous** may be compatible with certain cities and incompatible with others. Hazardous packages cannot originate nor pass through cities unless that city is compatible with hazardous packages (specified via the predicate **pc-compatible**).

Packages have a location and fees to be collected (specified via the predicates **at-package** and **fees-collected**, respectively). Hazardous packages require a permit and warning signs (specified via predicates **have-permit** and **warning-signs-affixed** predicates). Valuable packages require insurance (specified via the predicate **insured**).

4 Predicates

This section presents a summary of domain predicates. These may vary slightly or there may be additional predicates depending on the particular planner being used.

The following variables are used to indicate argument types in the list of predicates below:

Argument	*Type of Argument Value*
c	city
e	equipment
g	region
l	city-location
p	package
r	route
t	tcenter
v	vehicle

The following are the domain predicates: [5]

---

[5] These predicates may vary slightly or there may be additional predicates depending on the particular planner being used

Predicate	*Description*
at-equipment(e,l)	e is at l
at-package(p,x)	p is at x, a location or vehicle containing package
at-vehicle(v,x)	v is at x, a location or train (if vehicle is an attached traincar)
available(x)	x — a transport center, route, vehicle — is available
chute-connected(v)	chute of hopper truck/traincar v connected to (un)load cargo
clean-interior(v)	v has a clean interior
connects(r,o,d)	route r connects origin o to destination d. o and d are cities if r is an road route; airports if r is an air route; train stations if r is a rail route.
decontaminated-interior(v)	v has decontaminated/uncontaminated interior
door-open(v)	door open on v, an airplane, or a regular livestock truck/traincar
empty(x)	crane x is empty
fees-collected(p)	fees have been collected for package
guard-inside(v)	guard is posted inside armored truck/traincar v
guard-outside(v)	guard is posted outside armored truck/traincar v
have-permit(p)	permit has been obtained for hazardous package p
hose-connected(v,p)	hose connected from tanker truck/traincar v to (un)load cargo
in-city(l,c)	l is located in c
in-region(c,g)	c is located in g
insured(p)	insurance has been obtained for valuable package
pc-compatible(c,hazardous)	c is compatible with hazardous packages
pv-compatible(y,z)	package subtype y is compatible with vehicle (sub)type z
ramp-connected(x,v)	plane ramp x is connected to airplane v
ramp-down(v)	loading ramp down on v, a livestock or auto truck/traincar
rv-compatible(y,z)	route type y is compatible with vehicle type z
serves(t,x)	t serves x, a region (iff t is a hub) or a city
trough-full(v)	livestock trough full of food/water on livestock truck/traincar v
type(x,y)	individual x has type y (a type constant)
valve-open(v)	loading valve open on tanker truck/traincar v
warning-signs-affixed(p,v)	hazardous material warning signs have been affixed to vehicle v

5 Actions

This section describes the symbols that denote actions in UM Translog domain. Most symbol names are chosen to be self explanatory. For details, see the action definitions (available from the WWW site or via FTP) for the particular planner to be used.

5.1 Administrative Actions

Prior to carrying a package to its destination, fees should be collected, a special permit should be obtained if the package is of type **hazardous**, the package should be insured if it is of type **valuable**, and all these should be cancelled once the package is delivered at its destination. These activities are denoted by the action symbols **obtain-permit(?p)**, **collect-fees(?p)**, **collect-insurance(?p)**, and **deliver(?p)**, where *?p* is a variable symbol denoting a package.

5.2 Actions for loading/unloading

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

Loading a *regular* package into a *regular* vehicle involves opening the door of the vehicle, putting the package in, and then closing the door, denoted by the actions **open-door(?v)**, **load-package(?p ?v ?l)**, **close-door(?v)**. Unloading a regular package involves the same steps in reverse order, replacing load-package with **unload-package(?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.

Packages of type *valuable* can be carried only by vehicles of type *armored*, and require the additional steps of posting a guard outside while loading, and posting a guard inside while in transportation and removing the guards afterwards are required. These are denoted by **post-guard-outside(?v)**, **post-guard-inside(?v)**, and **remove-guard(?v)**.

Packages of type *hazardous* can be carried only with proper warning signs on the vehicle, and the vehicle must be decontaminated afterwards. These actions are denoted by **affix-warning-signs (?v)**, **remove-warning-signs (?v)**, and **decontaminate-interior(?v)**.

Loading/unloading a truck or traincar of type *flatbed* requires use of a crane denoted by *?c* in the actions **pick-up-package-ground(?p ?c ?l)**, **put-down-package-ground(?p ?c ?l)**, **pick-up-package-vehicle(?p ?c ?v ?l)**, and **put-down-package-vehicle(?p ?c ?v ?l)**.

Loading a truck or traincar of type *hopper* involves the actions **connect-chute(?v)**, **fill-hopper(?p ?v ?l)**, and **disconnect-chute(?v)**. Unload is similar, simply replace **empty-hopper(?p ?v ?l)** with **fill-hopper(?p ?v ?l)**.

Loading/unloading vehicles of type *tanker* involves the actions **connect-hose(?v)**, **disconnect-hose(?v ?p)**, **open-valve(?v)**, **close-valve(?v)**, **fill-tank(?v ?p ?l)**, and **empty-tank(?v ?p ?l)** in appropriate order.

Loading packages of type *livestock* involves the actions **lower-ramp(?v)**, **fill-trough(?v)**, **load-livestock(?p ?v ?l)**, and **raise-ramp(?v)**. Unloading involves the actions **lower-ramp(?v)**,

unload-livestock(?p ?v ?l), **raise-ramp(?v)**. **do-clean-interior(?v)**, and **unload-livestock(?p ?v ?l)**.

Loading/unloading packages of type *cars* involves the actions **load-cars(?p ?v ?l)**, and **unload-cars(?p ?v ?l)**. As in the case of livestock, the ramp of the vehicle needs to be lowered prior to loading/unloading, and the it should be raised immediately afterwards.

Loading/unloading vehicles of type *airplane* requires a conveyor ramp (denoted by $?r$) to be connected and the door of the vehicle to be open prior to the operation, and the ramp to be disconnected and the door to be closed afterwards. These activities are denoted by the actions **attach-conveyor-ramp(?v ?r ?l)**, **detach-conveyor-ramp(?v ?r ?l)**, **open-door(?v)**,**close-door(?v)**, **load-package(?p ?v ?l)**, and **load-package(?p ?v ?l)**.

5.3 Transit Actions

A vehicle $?v$ can be moved from its current location $?ol$ to another location $?dl$ if there is a route $?r$ of proper type between $?ol$ and $?dl$, using the action **move-vehicle(?v ?ol ?dl ?r)**. Vehicles of type *traincar* do not move by themselves but are pulled by vehicles of type *train* instead. Thus a traincar goes wherever the train it is attached to goes. The actions to attach/detach traincars to trains are **attach-train-car(?t ?c ?l)**, and **detach-train-car(?t ?c ?l)**.

In addition to those actions described above, UM Translog makes use of a dummy action called **do-nothing** which has no preconditions or effects.

6 Tasks

Tasks are the jobs to be planned for. Each task is either posed by the user to the planner as part of a planning problem, or it is introduced by a planner as subtasks if it serves to accomplish a task posed by the user. In this aspect tasks are similar to goals in STRIPS-style planning, only with richer structure. Figure 2 presents the organization of tasks and their subtasks, which are discussed below. For details, see the task/method definitions (available from the WWW site or via FTP) for the particular planner to be used.

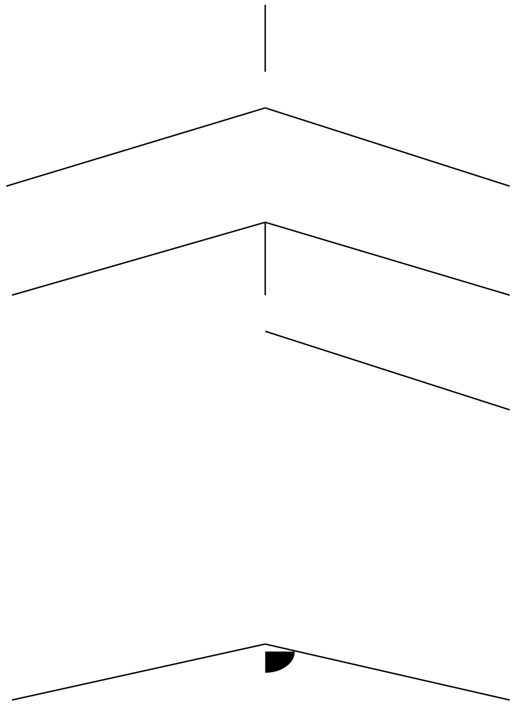

Figure 2: Top-level Task Hierarchy

AT-PACKAGE(?package ?destination) This task requires transporting the package to its destination.

TRANSPORT(?package ?origin ?destination) Provided that the package is currently in the origin location, this task involves picking up the package, carrying it to its destination, and delivering it.

PICKUP(?package) This task involves collecting fees, handling insurance and hazardous material permits. Insurance is required only for valuable packages, and permits are required only for hazardous packages.

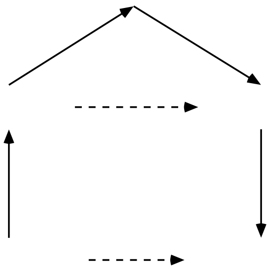

Figure 3: Transport Paths

CARRY(?package ?origin ?destination) This is the task of actually moving the 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. The diagram in Figure 3 shows the possible paths to transport a package. The *?origin* can be either *clocation1* (a customer location) or *tcenter1* (a transport center), and similarly the *?destination* can be either *clocation2* or *tcenter2*. As seen in the diagram, a package can be carried directly if there is a direct route available, otherwise it has to go through one or two transportation centers and possibly a hb. Transport within a city is termed "local" transport. Transport via a direct route (i.e., not involving a hub) is termed "direct" transport. Transport via a hub is termed "indirect" transport.

CARRY-DIRECT(?package ?origin ?destination) This task involves picking a route connecting the *?origin* and the *?destination*, and choosing a vehicle that is compatible both with the package, and the route. Only those vehicles that are at the *?origin* or one step away from the *?origin* can be dispatched. The task is accomplished by moving that vehicle to the origin, loading the package into the vehicle, moving the vehicle to the destination, and finally unloading the package.

AT-VEHICLE(?vehicle ?destination) If the vehicle is of type truck, plane, or train, it is moved to the destination, provided there is a direct route available from the current location. The type of the route must be suitable for the type of vehicle. If the vehicle is a train-car, a train must be moved to the location of the train-car, the train-car must be attached to the train, the train must be moved to the destination, and then finally, the train-car must be detached. Naturally, when a vehicle moves, so does the packages it contains.

LOAD/UNLOAD(?package ?vehicle ?location)

Loading and unloading involves issuing a sequence of actions to put the package into and out of the vehicle. The actions to be executed depends on the type of package and vehicle, and they were described in detail in Section 5.2. In particular, valuable packages can be transported only with armored vehicles, and they require guards posted outside while loading/unloading and guards inside, while in transit. Similarly, vehicles carrying hazardous packages need warning signs, which are removed after the package is unloaded and the vehicle is decontaminated.

A UM Nonlin Domain Definition

This section describes the definition of the UM Translog domain for UM Nonlin, a Common Lisp version of Tate's Nonlin planner. For details on how to use UM Nonlin, see the UM Nonlin user manual [4].

Documentation and code for UM Nonlin/UM Translog are available on the WWW at URL http://www.cs.umd.edu/plus/projects/Nonlin/ or by anonymous ftp from ftp.cs.umd.edu[6]

UM Translog operators for UM Nonlin are divided into 3 files that can be FTPed. In the *order* they are to be loaded, they are:

Set	*File*
Main Methods	umt-main-methods-nl.lisp
Load/Unload Methods	umt-load-methods-nl.lisp
Primitive Actions	umt-actions-nl.lisp

UM Nonlin has some idiosyncracies (described in its user manual) that may result in its not producing a plan in this domain (and others) or, in some cases, its producing an incorrect plan. This is more likely to occur when UM Nonlin is given a planning problem to solve with conjunctive goals, i.e. multiple packages to be delivered.

A.1 Nonlin-specific Domain Predicates

Because UM Nonlin does not handle negated predicates, additional predicates have been defined to specify (in method/action conditions and effects) the negation of other predicates. For a predicate p and its corresponding negative predicate q, whenever $p(x)$ is asserted for predicate p is made, then $q(x)$ should be retracted/deleted.

The additional predicates for the UM Nonlin are:

Predicate	*Opposite of Predicate*
chute-disconnected	**chute-connected**
dirty-interior	**clean-interior**
contaminated-interior	**decontaminated-interior**
door-closed	**door-open**
hose-disconnected(v)	**hose-connected**
ramp-disconnected(v)	**ramp-connected**
ramp-down	**ramp-up**
trough-empty	**trough-full**
valve-closed	**valve-open**
warning-signs-removed(v)	**warning-signs-affixed**

v is an argument of type vehicle. Except for predicates whose arguments are explicitly specified, a predicate has the same arguments are its opposite (see Section 4).

In order to specify that an individual does *not* have a specific type, the following negative type constants have been added in the UM Nonlin definition:

---

[6] For UM Translog, see /pub/nonlin/um-translog/README. For UM Nonlin, see /pub/nonlin/README.

Type Constant	*Opposite of*	*Assert for*
not-hazardous	**hazardous**	**package**
not-valuable	**valuable**	**package**
not-traincar	**traincar**	**vehicle**
not-tcenter	**tcenter**	**city-location**
not-hub	**hub**	**tcenter**

The "Assert for" column above specifies the type of individuals for which the type constant should be asserted for, if the opposite type does not apply. For example, the following would be asserted in the initial state for truck-1, a particular truck: `((type truck-1 truck), (type truck-1 not-traincar))`.

In UM Nonlin, packages with multiple subtypes need to have a composite package type specified for them, in addition to the individual types.

Package Composite Type	*Component Subtypes*
hazardous-t	hazardous, physical subtype t
perishable-regular	perishable, regular
perishable-liquid	perishable, liquid
valuable-t	valuable, physical subtype t
hazardous-perishable-liquid	hazardous, perishable, liquid
hazardous-perishable-regular	hazardous, perishable, regular
hazardous-valuable-t	hazardous, valuable, physical subtype t
perishable-valuable-liquid	perishable, valuable, liquid
perishable-valuable-regular	perishable, valuable, liquid
hazardous-perishable-valuable-liquid	hazardous, perishable, valuable, liquid
hazardous-perishable-valuable-regular	hazardous, perishable, valuable, regular

For example, a valuable, regular package package-1 needs to have the following assertions made in the initial state: `(type package-1 regular), (type package-1 valuable), (type package-1 valuable-regular)`. The following are the composite types and their component subtypes for package.

Like packages, vehicles with multiple subtypes also need a composite vehicle type specified for them.

Vehicle Composite Type	*Component Subtypes*
hazardous-valuable-regular	hazardous valuable regular
refrigerated-regular	refrigerated regular
refrigerated-tanker	refrigerated tanker
armored-regular	armored regular
armored-flatbed	armored flatbed

Composite package and vehicle types are used in assertions for the **pv-compatible** predicate in the always/initial state (see Section A.3 for an example of this).

A.2 Specifying the Problem

This section describes what needs to be initial state (context) for a UM Translog planning problem. For each individual x, various assertions need to be made depending on the (sub)types of x.

Listed by (sub)type below are the predicates for which assertions must be made in the initial state ("Required") and those for which assertions could be made ("Optional").

All of the applicable (sub)types should be consulted to determine required/optional assertions. For example, if truck-1 is a **regular**, **armored truck**, the *required* assertions for objects of type **vehicle**, **truck**, **regular** truck, *and* for **armored** truck must be made in the initial state: i.e., `(type truck-1 truck), (type truck-1 not-traincar), (type truck-1 regular), (type truck-1 armored), (at-vehicle truck-1 some-location), (door-open truck-1), (guard-inside truck-1).` If truck-1 is available to be used in the initial state, an assertion should be made for the (optional) vehicle preciate **available**: i.e., `(available truck-1).`

In the following lists of predicates, a "*" (kleene star) indicates that one or more assertions for a predicate are possible. For example, a tcenter airport-1 may serve multiple cities: city1, city2, etc. Thus one would have several assertions: `(serves airport-1 city1), (serves airport-1 city2), etc..`

In the following lists of predicates, a "/" (alternation) indicates that two or more alternatives, only one of which should be chosen. For example, If route-1 is an route, one should assert: `(type route-1` y`)`, where y is one of **road-route**, **rail-route**, or **air-route**.

Examples of actual problem definition can be found in section that follows.

A.2.1 Locations

region Required: (type x region).

city Required: (type x city), in-region. Optional: (pc-compatible x hazardous).

city-location Required: in-city.

tcenter Required: (type x tcenter), (type x hub / not-hub), (type x airport / train-station), in-city, serves*. Optional: available.

clocation Required: (type x clocation), (type x not-tcenter).

post-office Required: (type x post-office), (type x not-tcenter).

A.2.2 Routes

route Required: (type x road-route / rail-route / air-route), connects. Optional: available.

A.2.3 Vehicles

vehicle Required: at-vehicle. Optional: available.

airplane Required: (type x airplane), (type x not-traincar), door-open / door-closed.

train Required: (type x train), (type x not-traincar), door-open / door-closed.

truck Required: (type x truck), (type x not-traincar), (type x *physical subtype*). Optional: (type x *specialty type*)*, decontaminated-interior / contaminated-interior, warning-signs-affixed / warning-signs-removed.

traincar Required: (type x traincar), (type x *physical subtype*). Optional: (type x *specialty type*)*, decontaminated-interior / contaminated-interior, warning-signs-affixed / warning-signs-removed.

armored truck/traincar Required: guard-inside, guard-outside.

auto truck/traincar Required: ramp-down / ramp-up.

hopper truck/traincar Required: chute-connected / chute-disconnected.

livestock truck/traincar Required: clean-interior / dirty-interior, door-open / door-closed, ramp-down / ramp-up, trough-full / trough-empty.

regular truck/traincar Required: door-open / door-closed.

tanker truck/traincar Required: valve-open / valve-closed, hose-connected / hose-disconnected.

A.2.4 Equipment

equipment Required: at-equipment.

crane Required: empty.

plane-ramp Required: ramp-connected / ramp-disconnected.

A.2.5 Packages

package Required: (type x package), (type x *physical subtype*), at-package. Optional: (type x *specialty type*)*, fees-collected.

non-hazardous package Required: (type x not-hazardous). Optional: have-permit, warning-signs-affixed.

non-hazardous package Required: (type x not-valuable). Optional: insured.

A.3 A Sample Problem

The following illustrates the specification of an actual domain problem to UM Nonlin. The input to UM Nonlin consists of the always context forms, initial context (state) forms, and goal forms. The initial state is quite large due to the number of predicates that must be specificed for individual domain objects (vehicles, locations, etc.). We plan to make available our Problem Specifier code available that creates initial state descriptions from CLOS objects defined by the user or generated by our random Problem Generator.

This sample problem is a simple one: a regular package (pkg-1) is delivered from a customer location (city1-cl1) to another customer location (city1-cl2), in the same city (city1). The following is the input to Nonlin:

A.3.1 Always Context Forms:

```
((rv-compatible air-route airplane) (rv-compatible rail-route
 traincar) (rv-compatible rail-route train) (rv-compatible road-route
 truck) (pv-compatible hazardous-valuable-granular armored-hopper)
 (pv-compatible valuable-granular armored-hopper) (pv-compatible
 hazardous-granular armored-hopper) (pv-compatible hazardous-granular
 hopper) (pv-compatible hazardous-perishable-valuable-liquid
 armored-refrigerated-tanker) (pv-compatible
 perishable-valuable-liquid armored-refrigerated-tanker)
 (pv-compatible hazardous-valuable-liquid armored-refrigerated-tanker)
 (pv-compatible hazardous-valuable-liquid armored-tanker)
 (pv-compatible hazardous-perishable-liquid
 armored-refrigerated-tanker) (pv-compatible
 hazardous-perishable-liquid refrigerated-tanker) (pv-compatible
 valuable-liquid armored-refrigerated-tanker) (pv-compatible
 valuable-liquid armored-tanker) (pv-compatible perishable-liquid
 armored-refrigerated-tanker) (pv-compatible perishable-liquid
 refrigerated-tanker) (pv-compatible hazardous-liquid
 armored-refrigerated-tanker) (pv-compatible hazardous-liquid
 refrigerated-tanker) (pv-compatible hazardous-liquid armored-tanker)
 (pv-compatible hazardous-liquid tanker) (pv-compatible
 hazardous-valuable-bulky armored-flatbed) (pv-compatible
 valuable-bulky armored-flatbed) (pv-compatible hazardous-bulky
 armored-flatbed) (pv-compatible hazardous-bulky flatbed)
 (pv-compatible hazardous-perishable-valuable-regular
 armored-refrigerated-regular) (pv-compatible
 perishable-valuable-regular armored-refrigerated-regular)
 (pv-compatible hazardous-valuable-regular
 armored-refrigerated-regular) (pv-compatible
 hazardous-valuable-regular armored-regular) (pv-compatible
 hazardous-perishable-regular armored-refrigerated-regular)
 (pv-compatible hazardous-perishable-regular refrigerated-regular)
 (pv-compatible valuable-regular armored-refrigerated-regular)
 (pv-compatible valuable-regular armored-regular) (pv-compatible
 perishable-regular armored-refrigerated-regular) (pv-compatible
 perishable-regular refrigerated-regular) (pv-compatible
 hazardous-regular armored-refrigerated-regular) (pv-compatible
```

```
 hazardous-regular refrigerated-regular) (pv-compatible
 hazardous-regular armored-regular) (pv-compatible hazardous-regular
 regular) (pv-compatible hazardous-regular airplane) (pv-compatible
 regular armored-refrigerated-regular) (pv-compatible regular
 refrigerated-regular) (pv-compatible regular armored-regular)
 (pv-compatible regular regular) (pv-compatible regular airplane)
 (pv-compatible bulky armored-flatbed) (pv-compatible bulky flatbed)
 (pv-compatible liquid armored-refrigerated-tanker) (pv-compatible
 liquid refrigerated-tanker) (pv-compatible liquid armored-tanker)
 (pv-compatible liquid tanker) (pv-compatible granular armored-hopper)
 (pv-compatible granular hopper) (pv-compatible mail mail)
 (pv-compatible mail airplane) (pv-compatible livestock livestock)
 (pv-compatible auto auto))
```

A.3.2 Initial Context Forms:

```
((at-equipment ramp4 region1-ap1) (at-equipment ramp3 city2-ap1)
(at-equipment ramp2 city3-ap1) (at-equipment ramp1b city1-ap2)
(at-equipment ramp1a city1-ap1) (at-package pkg-1 city1-cl1) (at-vehicle
truck-1 city1-cl1) (available truck-1) (available road-route-i1656)
(available road-route-i1655) (available ramp4) (available ramp3) (available
ramp2) (available ramp1b) (available ramp1a) (available rail-route-4)
(available rail-route-3) (available rail-route-2) (available rail-route-1)
(available air-route-4) (available air-route-3) (available air-route-2)
(available air-route-1) (available road-route-2) (available road-route-1)
(available region1-ts1) (available region1-ap1) (available city3-ts1)
(available city3-ap1) (available city2-ts1) (available city2-ap1)
(available city1-ap2) (available city1-ap1) (available city1-ts2)
(available city1-ts1) (connects road-route-i1656 road-route city1 city2)
(connects road-route-i1655 road-route city2 city1) (connects rail-route-4
rail-route city3-ts1 region1-ts1) (connects rail-route-4 rail-route
region1-ts1 city3-ts1) (connects rail-route-3 rail-route city1-ts1
region1-ts1) (connects rail-route-3 rail-route region1-ts1 city1-ts1)
(connects rail-route-2 rail-route city1-ts1 city2-ts1) (connects
rail-route-2 rail-route city2-ts1 city1-ts1) (connects rail-route-1
rail-route city1-ts1 city1-ts2) (connects rail-route-1 rail-route city1-ts2
city1-ts1) (connects air-route-4 air-route city1-ap1 city1-ap2) (connects
air-route-4 air-route city1-ap2 city1-ap1) (connects air-route-3 air-route
city3-ap1 region1-ap1) (connects air-route-3 air-route region1-ap1
city3-ap1) (connects air-route-2 air-route city1-ap1 region1-ap1) (connects
air-route-2 air-route region1-ap1 city1-ap1) (connects air-route-1
air-route city1-ap1 city2-ap1) (connects air-route-1 air-route city2-ap1
city1-ap1) (connects road-route-2 road-route city2 city3) (connects
road-route-2 road-route city3 city2) (connects road-route-1 road-route
city1 city3) (connects road-route-1 road-route city3 city1) (door-closed
truck-1) (in-city region1-ts1 city3) (in-city region1-ap1 city2) (in-city
city3-ts1 city3) (in-city city3-ap1 city3) (in-city city3-cl1 city3)
(in-city city2-ts1 city2) (in-city city2-ap1 city2) (in-city city2-cl1
city2) (in-city city1-ap2 city1) (in-city city1-ap1 city1) (in-city
city1-ts2 city1) (in-city city1-ts1 city1) (in-city city1-cl2 city1)
(in-city city1-cl1 city1) (in-region city3 region1) (in-region city2
region2) (in-region city1 region1) (pc-compatible city3 hazardous)
```

```
(pc-compatible city2 hazardous) (pc-compatible city1 hazardous)
(ramp-available ramp4) (ramp-available ramp3) (ramp-available ramp2)
(ramp-available ramp1b) (ramp-available ramp1a) (ramp-disconnected ramp4)
(ramp-disconnected ramp3) (ramp-disconnected ramp2) (ramp-disconnected
ramp1b) (ramp-disconnected ramp1a) (serves region1-ts1 region1) (serves
region1-ap1 region1) (serves city3-ts1 city3) (serves city3-ap1 city3)
(serves city2-ts1 city2) (serves city2-ap1 city2) (serves city1-ap2 city1)
(serves city1-ap1 city1) (serves city1-ts2 city1) (serves city1-ts1 city1)
(type pkg-1 regular) (type pkg-1 not-valuable) (type pkg-1 not-hazardous)
(type truck-1 truck) (type truck-1 regular) (type truck-1 not-traincar)
(type ramp4 plane-ramp) (type ramp3 plane-ramp) (type ramp2 plane-ramp)
(type ramp1b plane-ramp) (type ramp1a plane-ramp) (type region1-ts1
train-station) (type region1-ts1 tcenter) (type region1-ts1 hub) (type
region1-ap1 airport) (type region1-ap1 tcenter) (type region1-ap1 hub)
(type city3-ts1 train-station) (type city3-ts1 tcenter) (type city3-ts1
not-hub) (type city3-ap1 airport) (type city3-ap1 tcenter) (type city3-ap1
not-hub) (type city3-cl1 clocation) (type city3-cl1 not-tcenter) (type
city3 city) (type city2-ts1 train-station) (type city2-ts1 tcenter) (type
city2-ts1 not-hub) (type city2-ap1 airport) (type city2-ap1 tcenter) (type
city2-ap1 not-hub) (type city2-cl1 clocation) (type city2-cl1 not-tcenter)
(type city2 city) (type city1-ap2 airport) (type city1-ap2 tcenter) (type
city1-ap2 not-hub) (type city1-ap1 airport) (type city1-ap1 tcenter) (type
city1-ap1 not-hub) (type city1-ts2 train-station) (type city1-ts2 tcenter)
(type city1-ts2 not-hub) (type city1-ts1 train-station) (type city1-ts1
tcenter) (type city1-ts1 not-hub) (type city1-cl2 clocation) (type
city1-cl2 not-tcenter) (type city1-cl1 clocation) (type city1-cl1
not-tcenter) (type city1 city) (type region2 region) (type region1 region))
```

A.3.3 Goal Forms:

```
((at-package pkg-1 city1-cl2))
```

A.3.4 Plan Output by Nonlin:

```
((:ND11120 :PRIMITIVE (DO-NOTHING))
 (:ND11114 :PRIMITIVE (DO-NOTHING))
 (:ND11110 :PRIMITIVE (P-COLLECT-FEES PKG-1))
 (:ND11436 :PRIMITIVE (P-OPEN-DOOR TRUCK-1))
 (:ND11442 :PRIMITIVE (P-LOAD-PACKAGE PKG-1 TRUCK-1))
 (:ND11444 :PRIMITIVE (P-CLOSE-DOOR TRUCK-1))
 (:ND11448 :PRIMITIVE (P-MOVE-VEHICLE TRUCK-1 CITY1-CL1 CITY1-CL2
                       LOCAL-ROAD-ROUTE))
 (:ND11450 :PRIMITIVE (P-OPEN-DOOR TRUCK-1))
 (:ND11456 :PRIMITIVE (P-UNLOAD-PACKAGE PKG-1 TRUCK-1))
 (:ND11458 :PRIMITIVE (P-CLOSE-DOOR TRUCK-1))
 (:ND11108 :PRIMITIVE (P-DELIVER PKG-1)))
```

B UMCP Domain Definition

This section describes the definition of the UM Translog domain for UMCP, a hierarchical task network planner. For details on how to use UMCP, see the UMCP user manual [1].

Documentation and code for UMCP/UM Translog are available on the WWW at URL http://www.cs.umd.edu/plus/projects/UMCP/.

Note that neither UMCP, nor the UM Translog domain specification for UMCP have been fully tested at the time of this technical report. All those documents are being made available free of charge with no implied guarantees or promises of support.

UMCP supports negation, thus UMCP domain specification does *not* contain the additional negatory predicates Nonlin needs. Furthermore, UMCP also supports disjunction, hence in several places multiple Nonlin opschemas are encoded as a single UMCP method.

B.1 Problem Specification

The problem specification for the initial state is the same as in Nonlin, except for small variations in the syntax.

The problems UMCP solves are represented as task networks, rather than conjunctive goals as in Nonlin. Task networks have a richer structure compared to conjunctive goals, because they allow conditions on how and when each task must be performed. Here is an example task network:

```
[(step1 AT-PACKAGE(review5 ComputingReviews))
(step2 AT-PACKAGE(paper2 AIJ-editor))
(step3 AT-PACKAGE(robot5 IJCAI-95-conference-site))
(step4 AT-PACKAGE(robot5 UM-CS-Dept))
(ord step2 step3)∧(ord step3 step4)
∧ (between AT-PACKAGE(robot5 IJCAI-95-conference-site) step3 step4)]
```

This task network specifies four tasks: A review to be sent to *Computing Reviews*, a paper to be submitted to an AIJ editor, a robot to be sent to IJCAI conference site and back. It also specifies the conditions that the paper must be submitted before shipping the robot to the conference site, and that the robot must remain at the conference site until it is shipped back.

The UMCP algorithm is provably sound and correct, and its implementation should correctly handle all of the interactions in multi package problems for which UM Nonlin fails.

References

[1] K. Erol, J. Hendler, D. S. Nau, and R. Tsuneto. UMCP User Manual. Technical report, Institute for Advanced Computer Studies, University of Maryland, 1995.

[2] K. Erol. *HTN Planning: Formalization, Analysis, and Implementation.* PhD thesis, 1995.

[3] K. Erol, J. Hendler, D. S. Nau, and R. Tsuneto. A critical look at critics in htn planning. In *IJCAI-95*, 1995.

[4] Subrata Ghosh, James Hendler, Subbarao Kambhampati, and Brian Kettler. *UM Nonlin — A Common Lisp Implementation of Nonlin: User Manual.* University of Maryland at College Park, Department of Computer Science, 1.2.2 edition, February 1992.

[5] Brian P. Kettler. *Case-based Planning with a Massively Parallel Memory.* Doctoral dissertation, University of Maryland at College Park, Dept. of Computer Science, 1995. In preparation.

[6] Brian P. Kettler, James A. Hendler, William A. Andersen, and Matthew P. Evett. Massively parallel support for case-based planning. *IEEE Expert*, pages 8–14, February 1994.

[7] Manuela M. Veloso. *Learning By Analogical Reasoning in General Problem Solving.* PhD thesis, Carnegie Mellon University, School of Computer Science, 1992.

[8] D. McAllester, and D. Rosenblitt. Systematic nonlinear planning. In *Proc. AAAI-91*, 1991.

[9] J. Penberthy, and D. S. Weld. UCPOP: A Sound, Complete, Partial Order Planner for ADL *Proceedings of the Third International Conference on Knowledge Representation and Reasoning, October 1992*

[10] Austin Tate. Generating Project Networks In *Proc. IJCAI-77*, 1977. pp. 888–893.

[11] David Wilkins *Practical Planning: Extending the classical AI planning paradigm,* Morgan-Kaufmann, CA. 1988.

[12] R. E. Fikes, and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.