Automatic Synthesis and use of Generic Types in Planning

Derek Long and Maria Fox
Department of Computer Science,
University of Durham, UK
maria.fox@dur.ac.uk, d.p.long@dur.ac.uk

Abstract

Domain-independent planning concentrates on the general algorithmic issues raised in exploring a search space generated in finding sequences of state-transition functions between an initial state and a goal state. It is acknowledged that domain-dependent planning, in which features of specific domains are exploited in this search, offers opportunities for more efficient planning, but at the price of greater effort in the domain-encoding. The work presented in this paper is concerned with giving a domain-independent planner access to certain kinds of domain-specific heuristics without the need for additional domain encoding effort. This is achieved by *automatically* identifying *generic types* from STRIPS planning domain descriptions. Generic types are higher order types allowing the categorisation of domains (and components of domains) into domain classes, including the commonly occurring *transportation domain* class. We show how the generic type structure of domains can begin to be exploited to increase planner efficiency. An interesting property of the work described here is that domain components which would not easily be recognised, by the human, as transportation problems can turn out to have an underlying transportation character which can be exploited by the application of standard transportation domain heuristics. The analyses described here are completely planner-independent and contribute to an increasing collection of *pre-planning* analysis tools which help to increase performance of planners by decomposing and understanding the structures of planning problems before planners are applied.

Introduction

Many planning problems feature objects which are *mobile* and which traverse *networks* of locations in the process of *transporting* non-mobile *portable* objects between initial and goal locations. Logistics, Gripper, Ferry and others are examples of domains in which this character is explicit and obvious to the human domain designer. We call these *standard* transportation domains. Transportation is such a common feature of planning problems, either as a central or an incidental component, that it is remarkable that most domain-independent planners do not attempt to exploit the feature in improving performance. Probably the key reason why they do not is that to do so would require that the planner should recognise the applicability of appropriate heuristics. This would appear to require some further effort on the part of the domain engineer and this runs counter to much of the work in domain-independent planning which seeks to operate with the most unadorned declarative domain descriptions and place the burden of problem-solving firmly on the planning system itself. There are also examples of domain components which have an underlying transportation character but which are not recognizable to the human domain designer as transportation problems (even if other components of the same domain have an obvious transportation character). A specific example, the PaintWall domain, is considered below.

In this paper we present an extension to TIM (Fox & Long 1998), a program comprising a range of static domain analysis techniques, which is capable of uncovering the transportation elements of a wide variety of standard and non-standard transportation domains. This extension is integrated with our Graphplan-derivative planner (Blum & Furst 1995), STAN (Long & Fox 1999) and improves the efficiency of STAN by allowing it to selectively exploit heuristics suited to transportation domains.

We call these domain elements *generic types*. Generic types are higher order types, that is, types populated by *types* rather than by domain objects. Having recognised mobiles and maps we can use them to prune action instantiations and to identify certain classes of unsolvable goals without search. The ability to infer generic types allows STAN to infer the relevance of a number of domain heuristics not normally available for exploitation by a planner. The potential benefits of exploiting such information have been illustrated by Kautz and Selman (Kautz & Selman 1998), who showed the performance advantages obtained by providing hand-coded transportation axioms to their SAT-based planners. Bacchus (Bacchus & Kabanza 1998) hand-codes similar domain-specific axioms in TLPlan. The hierarchical planner SHOP (Nau *et al.* 1999) relies upon

domain-specific control knowledge encoded in horn clause logic and used to supplement the declarative description of the domain.

Our analysis exploits the results of the basic type analysis performed by TIM and can be seen as an extension of the analysis from basic to generic type inference. We are currently able to infer the existence of mobiles, portable objects, carriers for portable objects and maps of locations accessible to mobiles and carriers. The advantage of the analysis performed by TIM is that it requires no pump-priming by the domain designer. The domain description is a standard STRIPS description which requires no annotation or other means of identifying characteristic features. The identification of the relationships that imply the generic type of an object is independent of the predicate and operator names used in the domain encoding, and is robust to the ordering of arguments within predicates. Although a domain designer will often encode a standard transportation domain using indicative predicate and object names, such as *at*, which suggests situatedness, and *holding*, or *in*, which suggest conveyance, the use of suggestive names cannot be relied upon for the automatic extraction of such structure from a domain description. Many transportation domains are non-standard in the sense outlined above, and complex domains may contain transportation sub-components, which even the domain designer has not recognised, or highly interconnected ones which require a number of unobvious predicates to encode.

The results of our analysis can be used in several ways by a planner. It is possible to use the generic types to filter operator instantiations by excluding instantiations which relate mobile objects to locations on maps which they cannot traverse or which relate portable objects to mobiles which cannot be carriers for the particular portable. These instantiations could not be excluded using basic types alone. The generic types make dependencies explicit which are not visible at the basic type level. Using basic types it is possible to exclude instances which require objects of type *truck* to be situated at objects of type *package* (obviously not well-typed), but not to eliminate instances which require trucks to be situated at locations which are not accessible to them. The analysis can also be used to detect unsolvable problems before attempting to plan for them. For example, a goal which places a mobile object at a location which is unreachable for that type of mobile, or which will give rise to sub-goals that do so, can be identified as unsolvable. We can also use the analysis to identify situations in which certain domain-specific heuristics can be exploited without loss of completeness. For example, the standard transportation heuristic which states that an object should never be collected from its destination or deposited at its start location (both BLACKBOX (Kautz & Selman 1998) and TLPlan(Bacchus & Kabanza 1998) exploit this in the Logistics domain) can lead to incompleteness if used in any transportation domain in which objects can play

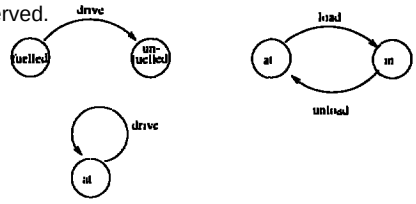

Figure 1: FSM depicting state transitions in the Rocket domain

roles other than simply being moved from one place to another. For example, keys in the Grid world are required for opening doors as well as being the objects that are moved between locations. The heuristic *can* safely be used if it can be inferred that the domain has a *pure* transportation character - that is, there are no features of the mobiles or the portables that could interfere with the simple transportation goals of the domain. We have experimented with the use of generic types to reduce the number of action instances and to reduce search where it can be inferred that the structure of the domain safely allows it. The results of these experiments are reported below.

The Basic TIM Analysis

The analysis described in this paper exploits the results of the basic type analysis performed by TIM (Fox & Long 1998). TIM automatically infers a rich type structure and a collection of invariants from an unadorned STRIPS domain description. In (Fox & Long 1998) it is shown that the inferred type structure enables STAN to prune action instances and obtain a significant speed-up in the plan generation process.

In the basic analysis of TIM a domain is viewed as a collection of Finite State Machines (FSMs) with domain constants traversing the states within them. States correspond to collections of *properties*. A property is a projection of a proposition onto one of its arguments, denoted by the predicate name subscripted with the argument position being considered. For example, in the Rocket domain (Blum & Furst 1995) there are rockets and packages, with rockets being capable of being at_1 locations and of moving, by driving, from being at_1 one location to being at_1 another, and of being $fuelled_1$ or $unfuelled_1$, and of moving between these two states. at_1 can be seen as forming a one-node FSM, and $fuelled_1$ and $unfuelled_1$ as forming a two-node FSM. This view is depicted in Figure 1.

Packages can be at_1 locations or in_1 rockets, and can move between these states in the resulting two-node FSM. In this example, rockets can be in states that involve more than one FSM, since they can be both at_1 and $fuelled_1$, or at_1 and $unfuelled_1$. The predicates in the initial state, and on the add and delete lists of operator schemas, provide TIM with a way of associating domain constants with FSM transitions, and are used

to form *property* and *attribute spaces* which partition the domain constants into a type structure. Each property space describes the properties that are exchanged during the state transitions that can be made by the associated domain objects. The exchanges of properties that are possible are described by *rules* included in the property spaces. A rule is of the form *enablers* $\Rightarrow$ *start* $\rightarrow$ *end* where the *enablers* are properties an object must have to be able to make the transition and *start* and *end* are the properties lost and gained in the exchange, respectively. For example, the rule $fuelled_1 \Rightarrow at_1 \rightarrow at_1$ is generated in the Rocket domain. Attribute spaces describe the properties that are acquired or lost, without exchange, when an object makes a state transition. For example, a location loses the property of having an object situated at it when that object moves, but it gains no other property in exchange for the one lost. The construction and use of property and attribute spaces is fully described in (Fox & Long 1998).

The property and attribute spaces provide the basis for the inference of a collection of invariants which can be used to reduce effort during the searching phase of plan generation. They also provide the foundation for the extended analysis described in this paper.

The Extended Analysis

The extended analysis performed by TIM involves the identification of mobile objects and the maps of locations upon which they move. It also identifies *portable* objects and *carriers* for those portable objects. We now describe the processes by which these generic types are inferred.

Inferring the Existence of Mobiles

The first phase in our analysis is concerned with identifying mobile objects. A mobile object is defined to be one that can make a self-propelled transition from being situated in one location to being situated in another location. Any domain description that contains mobiles will include a predicate for expressing the current situation of a mobile. For example, the predicate *at* might be used to express the fact that *truck1* is situated at *city1-2* in a standard Logistics encoding. The fact *at(truck1,city1-2)* expresses the situation of an explicit mobile object, the truck. We call the predicate in this fact an *at-relation*, because it expresses situatedness, but the name of the predicate is not important. For example, in the Mystery domain (McDermott 1998) (which is an encoded transportation domain) the fact *craves(rest,flounder)* expresses the situation of the mobile *rest* at the location *flounder*[1].

It might seem obvious that *at(truck1,city1-2)* expresses the situation of an explicit mobile object, and even that *craves(rest,flounder)* does, once the Mystery domain is decoded. However, there are domain behaviours that the human observer would not characterise as being of the transportation type and yet which can be revealed to be analogous to the standard transportation domains. An example is the PaintWall domain, consisting of several decorators, walls and paints. Figure 6 gives the two domain operators: a *paint* operator allows walls to be painted from one colour to another and a *go_to_wall* operator allows a decorator to move from one wall to another. A set of constraints determines which colours can be applied on top of which other colours. For example it is generally necessary for a wall to be primed and undercoated before it can be painted in the desired colour. Light paints cannot be placed over dark ones and some surfaces require paint-stripping before any other colours can be applied. There might be surfaces which, once applied, can never be removed. To complicate matters, not all paints need be suited to all walls. This feature of the domain, which can be found in the simple example of an initial state for the domain in Figure 7, requires careful examination to determine.

At first sight PaintWall might appear to be a standard transportation domain in which the decorators are mobile and the walls form the locations between which they move. However, this observation alone would not provide the planner with any exploitable structure, since the network of walls is completely connected (a decorator can move freely between any pair of walls) so there is no action pruning or search reduction implied. However, our analysis reveals that in fact the *walls* are mobile, traversing a network of locations formed by the paints! The edges in this network are given by the accessibility relation between pairs of paints. The analysis is also able to recognise the presence of mobile objects moving on a totally connected network of walls. STAN can exploit the detection of the walls as mobile very effectively, as our results demonstrate. The primary advantage is that STAN can avoid constructing action instances which would paint walls with unsuitable colours. The worked example below shows how this is done. As this example shows, the generic type analysis of a domain can assist the planner in distinguishing meaningful sub-types (in this case two sub-types of walls) and their associated behaviours within that domain. Similarly, in Logistics, our analysis can distinguish the sub-type of trucks moving on locations in *city1* from the sub-type of trucks moving around *city2*.

The starting point for the detection of mobiles is the identification of property spaces denoting FSMs comprising self-connected nodes formed from single properties. For example, in the FSM in Figure 2 the component denoting the at_1 property exchange reveals the presence of mobile objects in the collection of domain objects associated with the corresponding property space. TIM performs *sub-space analysis* to identify the subset of objects, in the property space, that can traverse this FSM component. Actually all of the ob-

[1] In the Mystery domain pleasures (including *rest*) encode robots, while foods (including *flounder*) encode locations. The *craves* predicate encodes situatedness while *feast* is the move operator and *eats* encodes accessibility between locations.

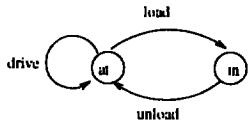

Figure 2: FSM depicting state transitions of cars and transporters: cars may be loaded into transporters and carried, or may drive separately.

jects can make the $at_1 \rightarrow at_1$ transition, but only the cars can make the $at_1 \rightarrow in_1, in_1 \rightarrow at_1$ transitions. As we explain below cars can be inferred to be both mobile and portable objects because of being able to make both kinds of transition. The transporters can be inferred to be the carriers for the cars.

In the description so far we have assumed that the at-relation, indicating the presence of mobiles, is a two-place predicate in which one of the arguments is the mobile itself and the other is the location at which it is situated. Although this is a limiting assumption it is a reasonable one and true of all of the benchmark domains. Our current failure to detect other, more complex, mobiles (such as ones for which the at-relation has more than two arguments, for example *at(robot,room1,fuel2)*, encoding the situatedness of a mobile with a given fuel level) does not compromise the soundness or utility of our existing analysis. We are also able to recognise the presence of *implicit* mobiles, encoded in predicates. For example, in Gripper, the fact *at_robby*(*room*1) refers to an implicit mobile (the robot) and the location of that robot at *room*1. Our analysis recognises one-place predicates as potential at-relations and is able to report the presence of implicit mobiles.

Figure 3 gives a pseudo-code description of the mobile-identification part of our analysis in property spaces and sub-spaces. The key observation regarding sub-spaces is that this analysis refines the analysis of the parent spaces, identifying subsets of mobile objects amongst a larger collection in the parent space.

Inferring the Maps Associated with the Mobiles

Mobiles traverse *maps* consisting of connected locations. Each mobile type uses a *move* schema to perform the transition. The name of the schema is not significant: whichever schema produces a $at \rightarrow at$ rule is identified as a *move* schema for that mobile type. The locations they can visit depend on the mobile type. For example, in Logistics encodings, the mobile type *truck* can traverse the roads linking locations in their respective cities, whilst the mobile type *airplane* can traverse the flight paths linking airports. Furthermore, in standard Logistics encodings trucks can only traverse the city networks that are accessible to their initial location. Part of the process of identifying mobiles is identifying

```
for each property space, P
    for each rule in P, r
        if r is of the form e ⇒ p → p
            where p is a property that appears in a singleton
                state in P
            and p corresponds to an arity 2 predicate, pred
        then construct a new mobile collection, M;
            associate r with M;
            put the objects in P into M;
            mark the other argument of pred as a location type
            record pred as the "at" relation of P;

for each subspace, S
    for each rule in S, r
        if r is of the form e ⇒ p → p
            where p is a property that appears in a singleton
                state in P
            and p corresponds to an arity 2 predicate, pred
        then if r is already associated with a mobile collection, M
            then if M is already refined
                then  contruct a related mobile collection, M';
                      associate M' with M;
                      put the objects in S into M';
                else  replace the objects in M with
                          objects from S;
                      mark M as refined;
            else construct a new mobile collection, M;
                 put the objects in P into M;
                 mark the other argument of pred as a location
                     type
                 record pred as the "at" relation of P;
                 mark M as refined;
```

Figure 3: Pseudo-code algorithm for detecting mobile objects in a planning domain description.

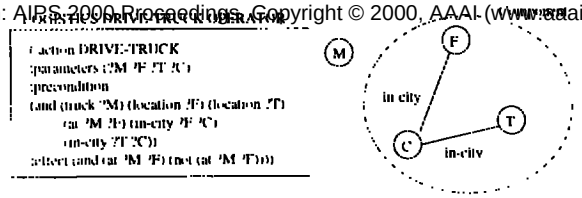

Figure 4: Connected component from the *drive_truck* operator in Logistics.

```
(:action TRAVEL
        :parameters (?T ?X ?Y)
        :precondition (and (borders ?X ?Y)
                           (at ?T ?X)
                           (has-visa ?T ?Y))
        :effect (and (at ?T ?Y)
                     (not (at ?T ?X))))
```

Figure 5: The *travel* operator

the maps they traverse. We do this by identifying *map links* in the preconditions of the relevant *move* schema. A map link is formed by one or more static connectivity relations indicating a direct link between two locations. The identification of maps involves the recognition of certain static relations as significant in determining the connectivity of the map.

The identification of maps relies upon the ability to determine the source and destination locations of a mobile transition. The source and destination of a transition are identified by finding the location argument in the at-relation on the preconditions and add-lists, respectively, of the *move* schema for that mobile. We construct a graph in which the nodes are the variables referred to by the operator schema and edges link pairs of variables that appear in the same preconditions in the schema. Having constructed the graph we identify the connected components which contain each of the source and destination variables. Collections of preconditions in the schema which refer only to variables in these connected components form map links in the location map traversed by the mobile. We exclude the mobile at-relation from the graph construction process, as we want to prevent the mobile itself from being used to identify map links except in the case where the map is dependent, in some way, on the mobile. Figure 4 indicates the result of map-inference in the Logistics domain. In this example the map-link is the proposition:

$$\exists C \cdot location(F) \wedge location(T) \wedge in\_city(F, C) \wedge in\_city(T, C)$$

and the map consists of all ways of satisfying this proposition for F and T. Map inference can be done in exactly the same way when the mobile itself is implicit in the operator schema.

An example of a domain in which a *dependent* map would be derived is the following Traveller domain. Travellers can only enter countries they have visas for, so that the map of countries associated with a traveller is dependent upon the visas held by that traveller. The *travel* operator in Figure 5 contains a *has_visa* precondition which enables this dependency to be detected by the map construction process. The relevant map-link is:

$$\exists T \cdot borders(X, Y) \wedge has\_visa(T, Y)$$

The map consists of all ways of satisfying this proposition for X and Y, and the question of whether a given traveller can traverse any specific link in the map is then determined by whether that traveller has a visa for the Y location.

Our current analysis has blindspots that we are currently addressing. First, our analysis would not, at present, recognise as mobile any object which had to pass through an intermediate state in the process of getting from a to b. For example, a frog which could only get from one riverbank to another by using a lilypad as a stepping-stone would not be seen as mobile. We use the presence of intermediate states to indicate that some third party is involved in the transportation of the object and that it is not, therefore, fully self-propelled. However, in this example, the lilypad is not in any sense transporting the frog. Secondly, the assumption that at-relations will always be one- or two-place is restrictive. It is possible to anticipate domain encodings in which an at-relation might also have arguments indicating levels of available resources, or other features. For example, the fact that an object is at a location and that that object has fuel available to it, might be wrapped up in a single predicate such as *at_with_fuel(object,location,boolean)*. We currently require this property to be broken down into the two distinct ones *at(object,location)* and *fuelled(object)*. Although this assumption does not seem unreasonable, and it is in fact the convention for STRIPS modelling, it is possible to see the mobile as moving in a multi-dimensional space defined by the other arguments to the at-relation.

A Worked Example of Mobile and Map Inference in the PaintWall domain

The PaintWall domain features the two operators in Figure 6. Given these, the basic analysis performed by TIM identifies the following FSMs. The processes by which these FSMs would be constructed is summarised in Section and fully described in (Fox & Long 1998).

The $painted_1$ FSM describes an unassisted state transition and is therefore recognised, by the extended analysis, as indicating the presence of an explicit mobile collection containing all of the objects that can make the transition from one $painted_1$ property to another.

```
(:action PAINTWALL
        :parameters (?D ?X ?F ?T)
        :precondition (and (painted ?X ?F)
                           (have ?T)
                           (can-cover ?T ?F)
                           (by-wall ?D ?X))
        :effect (and (painted ?X ?T)
                           (not (painted ?X ?F))))
(:action GO_TO_WALL
        :parameters (?D ?F ?T)
        :precondition (and (by_wall ?D ?F)
                           (wall ?T))
        :effect (and (by_wall ?D ?T)
                           (not (by_wall ?D ?F))))
```

Figure 6: The operators from the PaintWall domain

```
(define (problem paintwall)
(:init (have wood-undercoat) (have wood-primer)
       (have wood-white) (have wood-blue) (have wood-stripper)
       (have metal-primer) (have metal-undercoat)
       (have metal-white) (have metal-blue) (have metal-stripper)
       (can_cover wood-undercoat wood-primer)
       (can_cover wood-white wood-undercoat)
       (can_cover wood-blue wood-white)
       (can_cover wood-blue wood-undercoat)
       (can_cover wood-stripper wood-blue)
       (can_cover metal-undercoat metal-primer)
       (can_cover metal-white metal-undercoat)
       (can_cover metal-blue metal-white)
       (can_cover metal-blue metal-undercoat)
       (can_cover metal-stripper metal-blue)
       (wall kitchen-wall) (wall warehouse-wall)
       (by_wall decorator kitchen-wall)
       (painted kitchen-wall wood-blue)
       (painted warehouse-wall metal-blue))
```

Figure 7: A small initial state for the PaintWall domain.

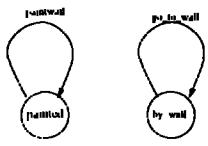

Figure 8: FSM depicting state transitions of walls and decorators.

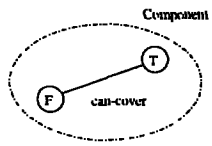

Figure 9: Connected component analysis from the *paint* operator in PaintWall. The nodes in the connected component are used to identify the map-link for map construction. The collection of preconditions in the schema (in Figure 6) refering to these nodes will be conjoined to form the map-link.

These are the objects of type *wall* – the only type associated with the property space defined by this FSM. The move operator for these mobiles is the *paint* operator which generated the rule $painted_1 \rightarrow painted_1$. Similarly, the decorators are identified as mobile on the map of walls (so walls are both locations and mobile on their own map).

The next stage in the analysis is to identify the maps of locations that the two mobile collections use. In the PaintWall case exclusion of the at-relation from the construction of the connected components reveals *can_cover* and *have* to be the only map-link predicates. Figure 9 shows the process by which the connected component, which forms the basis for the inference of the map traversed by the wall mobile, is constructed in the PaintWall domain. The inferred map-link is the proposition:

$$can\_cover(T, F) \wedge have(T)$$

and the map consists of all possible ways of satisfying this proposition by binding T and F. The map locations traversed by the walls are the paints. There are two components in the map defined by this relation for the example give in Figure 7. These components are isolated from one another – no wall can cross from one component of the map to the other. Our analysis sub-divides the walls into two mobile sub-types, one associated with each of the two components. Membership of the two sub-types is determined simply by path-existence from the starting locations of the walls in the two components. This observation can be used to dramatically reduce the number of action instances that would be built by a planner without recourse to this analysis. The network traversed by the decorators, which is inferred by analysis of the *go_to_wall* operator, is uninteresting in this sense since it is totally connected.

Inferring the Existence of Portable Objects and their Carriers

Transportation domains are characterised by the presence of objects which will be transported between locations by some, or all, of the mobiles in the domain. In a typical transportation domain there may be constraints that imply that only a subset of the mobiles

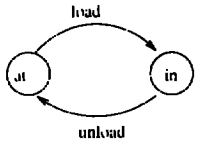

Figure 10: FSM indicating the presence of portable objects

can be carriers for particular kinds of portable objects (heavy objects might need to be transported by special carrying equipment, for example).

Our algorithm for inferring the existence of portable objects begins by looking for FSM structures which indicate that any transition in the location of associated objects must pass through an intermediate state involving a self-propelled (mobile) object. In Logistics, packages would be detected as portable because changes in their at_1 properties require them to pass through an *in-relation* with an mobile, indicating their transportation by a third party. Figure 10 shows the relevant FSM structure, inferred by TIM from the Logistics operator schemas. The first task is to identify the in-relation and establish the producing schemas for the $at \rightarrow in$ and $in \rightarrow at$ transitions. The producer for the $at \rightarrow in$ transition is the *load* schema and the producer for the $in \rightarrow at$ transition is the *unload* schema. The in-relation appears on the add-list of the *load* schema and on the delete-list of the *unload*, and it has both a portable argument and, in the case of explicit carriers, a carrier argument. We assume that portables are always explicit. Domain objects are made implicit by encoding them in predicates with corresponding dedicated operators. This is only feasible if there are few such objects, and whilst there is often only one mobile in a benchmark domain there are generally many portables.

It is possible for a collection of domain objects to be seen as both mobile and portable if that collection can perform both an unassisted state transition and an assisted one. This situation occurs for the cars in the domain described by the FSM in Figure 2. In Gripper the *pick* schema is inferred to be the *load* operator, and *drop* is inferred to be the *unload*. The in-relation is *holding*. In Ferry, *embark* and *debark* are the *load* and *unload* operators and *on* is the in-relation. In Mystery, *overcome* and *succumb* are the *load* and *unload* schemas, and *fears* is the in-relation. The existence of a two-state FSM of the form depicted in Figure 10 is not automatically indicative of the presence of portables. It is necessary for the at-relation to be associated with a location which appears on the map of some mobile that can be interpreted as the carrier for the portable and the in-relation to link the portable to a mobile. For instance, in a domain with lightswitches that can be turned on and off TIM would generate a similar two state FSM to that in Figure 10. However, the properties in the states would not be linked to locations and mobiles, so the lightswitches would not be seen as portable.

When portables have been extracted the identification of their associated carriers is quite straightforward. We identify as the carrier the mobile that is situated in the same location as the portable, in the case of the *load*, and in the destination of the portable in the case of *unload*, and which appears as the carrier argument to the in-relation. We do not currently deal with the case where multiple carriers are needed. If the carrier is implicit and the preconditions mention mobiles that must be present for loading or unloading to take place, then the analysis will not, at present, identify the carrier. For example, if a security guard must be present to witness the loading and unloading of the portable into, and out of, an implicit carrier then the carrier cannot be reliably identified. In all cases where the analysis cannot be definitive, it defaults, gracefully, to standard behaviour.

So that we can identify candidate carriers and the locations between which they move, mobile objects are inferred before portables in our algorithm. Space limitations prevent a fuller description of the algorithms: a detailed description is in preparation.

Results of Initial Application of the Generic Types Analysis

In this section we present the results of using the mobile extraction analysis to prune action instantiation in our Graphplan-based planner STAN. This represents only preliminary use of the analysis but illustrates the potential for benefit offered by this form of pre-planning analysis. All experiments were performed on a PII-300 PC with 128 Mb of RAM, running Red Hat Linux version 5.2. All timings are elapsed time measurements in milliseconds. We have presented results obtained from standard benchmark domains for ease of comparison.

The ability to relate different mobile types with the location maps upon which they can move enables a significant reduction in the number of well-typed action instances. For example, any attempt to move, load or unload a mobile object to, from or at a location not accessible to it, is not well-typed. The generic type inference is done before action instantiation, and informs the process of instantiation by associating, with each schema, the mobile objects from which the mobile argument in the schema can be instantiated and the locations accessible to these objects on their maps. If a schema has no mobile argument then either the mobile is implicit in that schema, in which case only the accessible locations are supplied, or the situation of mobiles is irrelevant to the schema, in which case STAN defaults to its standard instantiation process on that schema. In Logisitics, in which there are several disconnected maps traversed by different sub-types of the truck mobile, we have been able to solve problems

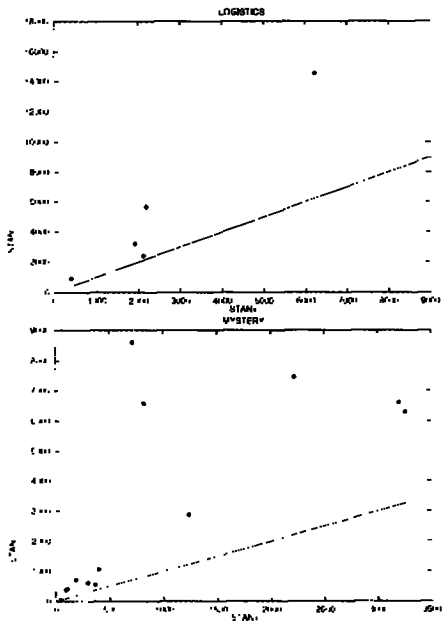

Figure 11: Results derived from Logistics and Mystery demonstrating a comparison between STAN and STAN+ (STAN with generic type inference) on AIPS-98 competition data sets.

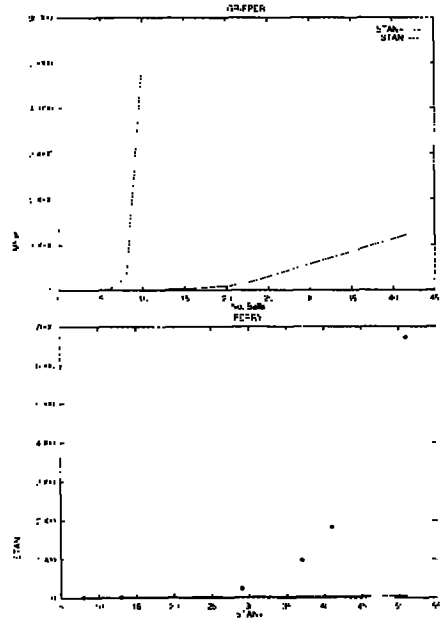

Figure 12: Results derived from Gripper and Ferry demonstrating a comparison between STAN and STAN+. These domains have implicit mobiles.

not previously solvable by STAN. Figure 11 illustrates the comparative performances of STAN without generic type analysis and STAN with this analysis (STAN+) on a collection of 6 Logistics problems from the AIPS-98 competition data set. In this graph the solid line indicates equal performance and points above the line indicate an advantage for STAN+. STAN+ was able to solve an additional problem, problem 17, in 8 seconds, which could not be solved with available resources by STAN. Problem 4 was solved by STAN+ in 25 seconds, and by STAN in 42 seconds, but this point was not included in the graph because it would have obscured the other data.

The recognition that the domain is a transportation domain, together with the observation that objects in the domain play no other role than to be transported, enables STAN+ to exploit a heuristic which prunes *load* instances, that load portables from their final destinations, and *unload* instances that unload portables at their initial situations. This is a powerful general heuristic which is one of the key performance-enhancing heuristics supplied by Kautz and Selman to SATPLAN in (Kautz & Selman 1998). Kautz and Selman supply this heuristic as a hand-coded, domain-specific, axiom. The heuristic exploited by STAN+ is domain-independent, in the sense that STAN+ can exploit it whenever it is able to infer the appropriate domain structure without explicit instruction from the domain engineer. All of the data sets illustrated here were generated using this heuristic, since they are all pure transportation domains. It must be emphasised that the decision to use the heuristic is made automatically, by STAN+, not by the user manually configuring the behaviour of the system.

The Gripper and Ferry domains both have implicit mobiles. In these domains there is no benefit to be obtained simply by excluding action instances that refer to unreachable locations because all locations are reachable by the mobiles. The dramatic performance benefits demonstrated by the graphs in Figure 12 derive from combining the symmetry machinery, described in (Fox & Long 1999) and implemented in both STAN and STAN+, with the heuristic described above. It should be noted that this combination allows STAN+ to solve optimally all 20 of the competition Gripper problems. In the competition only 4 of these problems were solved optimally by any of the planners. This explains why the data in Figure 12 is presented as two separate data plots for this domain, rather than as a single comparative plot as for the other domains. The Ferry domain has similar characteristics to Gripper and benefits from the same combination of analyses.

The PaintWall domain presented in this paper was given as an example of the potential for our generic types analysis in a non-standard transportation domain. The PaintWall instances in which the identification of mobiles yield an advantage are those in which the map, traversed by the mobiles, is split into two or more disconnected components. The following formula

Logistics			Mystery		
Prob.	With GT	No GT	Prob.	With GT	No GT
1	336	1368	1	102	186
2	946	3600	2	608	3680
3	2150	16156	3	608	1856
4	3210	26013	4	130	252
5	326	2205	5	552	3096
6	3671	72648	6	5544	14454
7	1678	8690	7	204	1128
8	3823	>66580	8	2430	6372
9	6273	62260	9	512	2408
10	6668	46160	10	3132	42174
11	1366	4680			
12	11559	99441			
13	13627	>45037			
14	7551	73216			
15	1536	10539			
16	5843	36751			
17	2624	19700			
18	24340	>89100			
19	14574	>79296			
20	20069	>87615			

Figure 13: Number of action instances generated by STAN with and without generic types analysis: where a lower bound is given the instantiations overflowed 512Mb of memory.

gives the exact number of action instances pruned when there are n map components, where the ith component has e_i edges and m_i mobiles that traverse it.

$$\sum_{i=1}^{n}(m_i . \sum_{j=1, j\neq i}^{n} e_i)$$

For example, in the instance presented in Figure 7 the number of actions pruned will be 10, out of the total action instance set of 22. Figure 13 shows the number of action instances generated by STAN with and without generic type analysis in a selection of instances from the AIPS 98 competition data set.

Related Work

The integration of domain-specific mechanisms into domain-independent planning has been considered in a few contexts: Srivastava and Kambhampati (Srivastava & Kambhampati 1999) show how a planner can exploit resources in the generation of plans. They observe that, as resources increase, planning becomes harder because of the corresponding increase in the number of ways to allocate those resources. Their work demonstrates that the separation of the resource allocation from the planning process improves the planning performance. However, this separation is not automatic as it relies on annotation of the domain description by hand.

Automatic pre-planning domain analysis has alos been previously explored by several researchers. Koehler (Koehler 1998) uses an automatic pre-planning analysis to assist IPP by identifying subproblems, representing milestones in the planning process, which can be ordered to avoid some search. However, this analysis renders IPP incomplete for optimal plans in some domains so cannot be safely exploited fully automatically. Nebel, Dimopoulos and Koehler (Nebel, Dimmopoulos, & Koehler 1997) provide IPP with a number of configurable options to allow the user to select filters (RIFO) which eliminate irrelevant facts and objects from the domain prior to instantiation of actions.

Fully automatic pre-planning analyses have also been explored. Porteous and McCluskey (McCluskey & Porteous 1997) have used automatic pre-planning analysis of domains encoded in their object-centred language, OCL, to identify goal orderings and macro-operator sequences. Gerevini and Schubert (Gerevini & Schubert 1996), Scholz (Scholz 1999), Kelleher and Cohn (Kelleher & Cohn 1992) and Fox and Long (Fox & Long 1998) carry out domain invariant extraction for the purpose of supplementing the reasoning processes of the planner. Refanidis (Refanidis & Vlahavas 1999) and Geffner and Bonet (Geffner & Bonet 1998) pre-process planning domains to extract search heuristic functions which they then use to inform a variation on the A* search strategy. Howe *et al.* (Howe *et al.* 1999) have begun to explore automatic planner selection on the basis of suitability to particular domain structures. This is based on a quantitative measurement of the extent to which a particular problem fits the profile of a particular planner, using a detailed empirical and statistical analysis of the candidate planners.

Conclusion

The work described in this paper extends the fully automatic strand of pre-planning analysis by stepping back from analysing the incidental properties of individual domains and instead looking for the defining characteristics of certain *classes* of domains. Classifying domains according to their characteristic features (such as the presence of transportation components) allows the automatic invocation of heuristics suited to the appropriate class of domains. We have shown how the classification of domains as pure transportation domains allows the use of a domain specific heuristic that dramatically improves the performance of the planner on these domains.

The identification of higher order types provides us with the potential to perform more precise type-checking than can be managed using the basic type structures inferred by TIM, allows us to invoke more powerful invariants than can currently be inferred to be relevant by any other automatic technology and to identify certain unsolvable goals without planning. We have so far shown that the results of our analysis can improve the performance of planners by enabling drastic pruning of incorrectly typed action instances and the elimination of search along paths which, because of the nature of the domain, can be inferred to lead necessarily to dead ends. We are currently working on extending

our analysis to capture more sophisticated notions of mobility and portability and considering new ways to exploit the information in the search process itself.

References

Bacchus, F., and Kabanza, K. 1998. Using temporal logic to express search control knowledge for planning. Technical report, University of Waterloo, Canada, ftp://logos.uwaterloo.ca/pub/bacchus/BKTlplan.ps.

Blum, A., and Furst, M. 1995. Fast Planning through Plan-graph Analysis. In *IJCAI.*

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *JAIR* 9.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Geffner, H., and Bonet, B. 1998. High level planning and control with incomplete information using POMDPs. In *Proceedings of AIPS-98 workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments.*

Gerevini, A., and Schubert, L. 1996. Accelerating Partial Order Planners: Some Techniques for Effective Search Control and Pruning. *JAIR* 5:95–137.

Howe, A.; Dahlman, E.; Hansen, C.; Scheetz, M.; and von Mayrhauser, A. 1999. Exploiting competetive planner performance. In *Proceedings of the Fifth European Conference on Planning, Durham, UK.*

Kautz, H., and Selman, B. 1998. The role of domain-specific axioms in the planning as satisfiability framework. In *Proceedings of AIPS-98, Pittsburgh, PA.*

Kelleher, G., and Cohn, A. 1992. Automatically Synthesising Domain Constraints from Operator Descriptions. In *Proceedings ECAI92.*

Kochler, J. 1998. Solving complex planning tasks through extraction of subproblems. In *Proceedings of AIPS-98, Pittsburgh, PA.*

Long, D., and Fox, M. 1999. The efficient implementation of the plan-graph in STAN. *JAIR* 10.

McCluskey, T. L., and Porteous, J. 1997. Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency. *Artificial Intelligence* 95(1).

McDermott, D. 1998. PDDL – the planning domain definition language. Technical report, Yale University, http://www.cs.yale.edu/users/mcdermott.html.

Nau, D.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Nebel, B.; Dimmopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proceedings of ECP-97, Toulouse, Fr.*

Refanidis, I., and Vlahavas, I. 1999. GRT: A domain independent heuristic for STRIPS worlds based on greedy regression tables. In *Proceedings of the Fifth European Conference on Planning, Durham, UK.*

Scholz, U. 1999. Action constraints for planning. In *Proceedings of the Fifth European Conference on Planning, Durham, UK.*

Srivastava, B., and Kambhampati, S. 1999. Planning though separate resource scheduling. In *AAAI Spring Symposium on Search Strategy under Uncertain and Incomplete Information.*