Testing Domain Model Validation Tools

M. M. West and D. E. Kitchin,
The School of Computing and Engineering,
The University of Huddersfield,
Huddersfield, UK

Abstract

If a planning domain model contains bugs and inconsistencies then no matter how efficient the planning algorithm, it is very likely that at least some of the plans produced will be flawed. We consider the familiar 'Tyres World' domain and compare its validation using two modelling languages and the tools which support them.

1 Introduction

The development of AI planning systems is difficult and error-prone. If the planning domain model contains bugs and inconsistencies then no matter how efficient the planning algorithm, it is very likely that at least some of the plans produced will be flawed. In consequence, validation is recognised as a critical task [9]. The problems associated with faulty planning domain models have been addressed by, for example, Grant [2], who also cites several other authors in their identification of faulty plans resulting from faulty planning domains. The emphasis of Grant, however, is on the faulty plans, which can be used for the subsequent correction of the domain. In contrast, our emphasis is in looking at the faulty domain model.

West and McCluskey have used machine learning techniques to validate a formal specification of an Air Traffic Control (ATC) domain (in the IMPRESS project–see for example [15]); the authors noted that there are similarities between the validation of requirements models and that of knowledge based systems. In this paper we will also be looking at the use of a formal specification language for domain model encoding. We consider the familiar 'Tyres World' domain [11] and compare its validation using two modelling languages and the tools which support them. Our reasons for choosing the Tyres domain model are that it is a well-known and well-used model that is unlikely to have any hidden errors. Our strategy is to introduce errors and see if the tools detect them.

The first language is OCL [7, 5] and its supporting tools environment GIPO[13, 6]. OCL is an object-centred domain modelling language. It is a structured language that allows models to be constructed, statically and dynamically validated, and maintained with relative ease. As well as action descriptions, an OCL model also contains substate class definitions and other invariants. These capture information about valid states for objects and valid planning states. Invariants are also an important part of a specification written in the second language, the 'B' Abstract machine notation (B-AMN) [12]. In contrast, a domain model written in a domain description language like PDDL [1] does not usually contain invariant information, but focuses on action descriptions. GIPO is a GUI tool-supported environment for the creation of

AI planning domain models, incorporating a stepper, validation tools, and planning engines. The stepper allows the user to interact directly with the model, choosing and applying actions for a chosen task. The supporting tools environment for B-AMN is the B-Toolkit [4]. In [14] we identified the similarities between *state-based* formal specification languages and planning languages. In many languages (and in particular OCL) there is the notion of a state and of an operation which potentially alters the state and is defined using pre- and postconditions. In OCL a *task* for which we generate a plan is partly defined in terms of state initialisation and there is a similar initialisation of state in B-AMN. There is also a similarity between methods of validation – for example animation compares with the 'stepper' in GIPO and the discharge of proof obligations for consistency has some equivalences with consistency checks in GIPO.

The motivation for this paper is to investigate any correspondence between OCL and B-AMN, with the potential for automatic translation from B-AMN to OCL. Previous work described the use of a formal specification language (B-AMN) for the capture and validation of a planning domain model [14]. We identified as possible future work an investigation into the process of implementation, by which we meant the translation from the AMN specification into a language suitable for input to a planner. The B-Toolkit supports the processes of refinement and implementation of an abstract machine, but the target language would normally be a programming language such as C++ or Java. We need to test whether the translation from AMN to PDDL or OCL is straightforward. This paper describes our first steps towards answering that question. We also intend to investigate the adequacy of the validation tools. We began with a domain model of the Tyres world, written in OCL and a B-AMN specification of the same world[1]. The approach was the deliberate introduction of equivalent errors into each model to see if the use of the stepper/animator and validation checks and proof tool, would identify these faults. This is described in more detail in section 2.

2 Tyres World: Strategy for Introducing Errors

The Tyres world domain involves the changing of a faulty wheel using a wrench and a jack, both of which are (usually) initially in the car boot. In the case of OCL, two wheels, hubs and their attached nuts were modelled plus a spare wheel in the boot. The additional wheel (as compared with the 'usual' model in [11]) was introduced so that extra validation checks could be introduced. In contrast, in the B-AMN model *four* wheels plus hubs and nuts were modelled. However, as it turned out, two would have been sufficient. A similar model was created in B-AMN using the same objects and the same operations[2]. The B-AMN model we used was, as far as we knew, 'correct'. In both application areas (functional requirement acquisition and knowledge engineering) it is never possible to formally prove that the 'real-world' has been correctly modelled. However validation tools help us to *challenge* some of our assumptions [10].

Our experiments both *challenged* the two models and also tested the two validation tools in their capabilities of providing such a challenge. The planning model of the Tyres World has been intensively validated so we should expect to find few or no errors in the original model. For that reason errors were introduced (into the two models) as follows:

[1] These can be found at http://scom.hud.ac.uk/scommmw/TyresWld/

[2] An exception is the case where simplifications were possible in B-AMN such as the use of a single operation for 'fetching a tool'.

(i) Errors in the initial state - an invalid initial state;
(ii) Errors in preconditions and postconditions;
(iii) Errors in setting tasks (impossible goals).

The next paragraphs describe the errors and the following sections describe the results of validation for each of the two tools.

Errors in the initial state We introduced errors such as the obviously incorrect state of two wheels on a single hub.

Errors in pre- and postconditions The preconditions for each operation can be roughly divided into two kinds. The first kind is domain specific operations concerned with a specific object, for example the use of a wrench for loosening nuts, but which are not in general concerned with overall consistency. The second kind can be linked to a general rule, that a tool is not available for a task if it is in the boot for example. Errors of each kind were introduced and the results are demonstrated.

Errors were deliberately introduced into the postconditions for selected operations - for example after replacing the nuts on the wheel (operation *do_up*) one of the postconditions is that the nuts would be on the wheel and loose. This postcondition was removed.

Errors in setting tasks Errors were introduced in the OCL task description within GIPO. These involved an attempt to reach an illegal state (i.e. one which was inconsistent with the domain model).

3 Tyres World in B AMN: A Comparison

We now describe the tyres world in B-AMN and compare it with the OCL version. B-AMN is a modular language: a typical specification is composed of several 'abstract machines'. An abstract machine state comprises several variables which are constrained by a machine invariant and initialised. Operations on the state contain explicit preconditions; the postconditions are expressed as 'generalised substitutions', giving the language a 'program-like feel'. Machine composition is achieved by (for example) the INCLUDES mechanism which allows one machine to alter the data of another. This gives specifiers the opportunity of breaking down a large model into smaller components – however since Tyres World is small we used only *one* machine. B-AMN uses sets as a basis for its data type. The sets comprising *Wheel* consisted of $\{Wl1, Wl2, Wl3, Wl4, spare\_wl\}$, corresponding to the reduced set $\{wheel1, wheel2, wheel0\}$ in OCL. *Hub* in B-AMN is a set of 4 hubs whereas there are two hubs $(hub1, hub2)$ in OCL and *Wl_Nuts* is the set of 4 wheel nuts in B-AMN (whereas there are two in OCL).

There is also a set *Tool* which includes a jack and a pump and a set *Container* which includes a boot. Notice that in B-AMN we use *upper* case for sets and (as will be seen) for operations also. Otherwise the data types and operations as far as possible mimic those of the OCL domain model. In OCL (and PDDL) the states of the system are modelled by predicates. Thus in OCL, '$open(boot)$' means the boot is open and '$tight(nut1, hub1)$' that nut1 is tight on hub1. In contrast, in B-AMN data types are chiefly modelled by sets and set constructs. For example *Open* is a function between *Container* and the booleans {TRUE, FALSE} so the function value $Open(boot) = TRUE$ indicates the boot is open. Predicates of arity 2 in OCL are also modelled by functions in B-AMN so $Tight(nut1) = hub1$ models the fact that nut1 is

tight on hub1. It can be seen that there is a potential correspondence between OCL predicates of arity 1 and 2 with B-AMN partial functions. OCL predicates of arity one, $pred(x)$ map to B-AMN total functions whose *domain* is the type of x and whose *range* is the booleans. OCL predicates of arity 2, $pred(x, y)$, map to B-AMN partial functions whose *domain* is the type of x and whose *range* is the type of y.

In OCL, consistency checks are facilitated by the existence of 'substate classes' which describe the states inhabited by the various objects which are assumed to be distinct from each other. Thus nuts can be tight (on a hub) or loose (on a hub) or we could '*have_nuts*'. There are also 'inconsistent constraints' which forbid certain combinations of object substates (we return to this in Section 4.1). In B-AMN these constraints are modelled by an INVARIANT using logic and set theory. Thus the domain of function *Loose* (all loose nuts) merged with the domain of function *Tight* (all tight nuts) merged with the nuts for which $Hold\_Nuts(nuts) = TRUE$ equals the whole of the set *Wl_Nuts*. Also, these three sets are distinct (have zero intersection). These two constraints can be expressed as a predicate, part of which is shown here:

$\mathsf{dom}(\ Hold\_Nuts \rhd \{\ TRUE\ \}\ ) \cup \mathsf{dom}(\ Tight\ ) \cup \mathsf{dom}(\ Loose\ ) = Wl\_Nuts \wedge$
$\mathsf{dom}(\ Hold\_Nuts \rhd \{\ TRUE\ \}\ ) \cap \mathsf{dom}(\ Tight\ ) = \varnothing \wedge$
$\mathsf{dom}(\ Hold\_Nuts \rhd \{\ TRUE\ \}\ ) \cap \mathsf{dom}(\ Loose\ ) = \varnothing$
(etc.)

4 Introducing Errors in Tyres World (GIPO)

Various tasks were tried out in GIPO using both the stepper and planning engines to see if errors and inconsistencies in the domain model were detected, and to compare its performance with that of the BTool. GIPO does perform some useful validation checks. Checks on operators include ensuring that they consist of legal expressions. Checks on tasks include ensuring that initial states are fully instantiated and the goal is a legal substate expression as defined in the domain model. The effect of errors introduced is presented in the following subsections.

4.1 Inconsistent Initial State

(a) 2 wheels and 2 nuts on one hub

A deliberate error was introduced in the initial state - that two wheels were on one hub. The other hubs were in a legal state. The OCL for the goal and this incorrect state is presented next.

```
% Goals
   [
    se(wheel,wheel0,[wheel_on(wheel0,hub1)])],
   % INIT States
   [
    ss(container,boot,[open(boot)]),
    ss(nuts,nuts_1,[tight(nuts_1,hub1)]),
    ss(nuts,nuts_2,[tight(nuts_2,hub1)]),
    ss(wheel,wheel1,[wheel_on(wheel1,hub1)]),
    ss(wheel,wheel2,[wheel_on(wheel2,hub1)]),
```

```
    ss(wrench,wrench0,[have_wrench(wrench0)]),
    ss(wheel,wheel0,[have_wheel(wheel0)]),
    ss(hub,hub1,[on_ground(hub1),fastened(hub1)]),
    ss(pump,pump0,[pump_in(pump0,boot)]),
    ss(jack,jack0,[jack_in_use(jack0,hub0)]),
    ss(hub,hub0,[jacked_up(hub0,jack0),fastened(hub0)])]
```

As discussed above, additional objects were added to the original domain model, so that there were 2 hubs, 3 wheels and 2 sets of nuts. An inconsistent initial state was created with 2 wheels (*wheel*1 and *wheel*2) on the same hub (*hub*1) and 2 sets of nuts, both tight, on the same hub. *wheel*0 (the spare wheel) is available for use; *have_wheel*(*wheel*0) is true. The goal was to have the spare wheel on *hub*1.

GIPO did not object to this inconsistent initial state - no errors were found by the validation checks. When tried with the planners, one of them (FF [3]) reported that the goal was impossible. The other planner, a simple forward planner, failed to find a solution.

(b) Nuts on hub but no wheel

An initial state was created in which the nuts were on one of the hubs but no wheel was on that hub. The other hubs were in a correct state (as seen below).

```
[
    ss(wheel,wheel1,[wheel_in(wheel1,boot)]),
    ss(wheel,wheel2,[have_wheel(wheel2)]),
    ss(pump,pump0,[pump_in(pump0,boot)]),
    ss(jack,jack0,[have_jack(jack0)]),
    ss(hub,hub0,[on_ground(hub0),fastened(hub0)]),
    ss(container,boot,[closed(boot)]),
    ss(nuts,nuts_0,[tight(nuts_0,hub0)]),
    ss(wrench,wrench0,[have_wrench(wrench0)])]
```

This task is inconsistent - *wheel*1 is in the boot, we have *wheel*2, the nuts (*nuts*_0) are tight on *hub*0 and *hub*0 is on the ground and fastened. This implies there's a wheel on the hub, as the hub is not free, but as can be seen neither of the two wheel objects are on it.

Using the stepper we get to a state where there is no applicable operator. That is, we apply the appropriate operators: *open_boot*, *fetch_wrench*, *loosen*, *jack_up*, *undo_nuts* until we reach a state where the hub is unfastened and jacked up, but it is not free. This implies that there is a wheel on the hub. This is not the case, so we are unable to remove the non-existent wheel. Nor can we put one of the available wheels on the hub because that requires the hub to be free as well as unfastened and jacked up. Therefore there are no applicable operators which will allow us to achieve the goal.

This inconsistency is not found by the validation checks. The user can find it by using the stepper or using a planner that incorporates such checks: FF for example reports that the goal is impossible.

Our conclusion from these two experiments was that GIPO does not exploit all the validation checks made possible by the structure and content of the OCL. For example initial states (a) and (b) violate the 'inconsistent constraints' part of the OCL Tyres World domain model in that

```
inconsistent_constraint([wheel_on(W1,X),wheel_on(W2,X),ne(W1,W2)]).
inconsistent_constraint([free(X),tight(Nuts,X)]).
```

4.2 Missing Pre- and postconditions

(a) Removal of wrench condition:

Prevail conditions in OCL are preconditions that persist - that is, the object concerned does not change state during the operation. An example of this would be the prevail condition *have_wrench*(W) of the *loosen* operator:- in order to loosen the nuts we must have the wrench - and we will still have the wrench after the nuts have been loosened. We removed the *have_wrench*(W) prevail condition from the *loosen* operator.

```
operator(loosen(W,H,N),
    % prevail with    se(wrench,W,[have_wrench(W)]) removed
    [
     se(hub,H,[on_ground(H),fastened(H)])],
    % necessary
    [     sc(nuts,N,[tight(N,H)]=>[loose(N,H)])],
    % conditional
    []).
```

This error, as expected, was not detected by validation checks, but became apparent when using GIPO's stepper. The operator was not able to be applied because the wrench was not available. This type of error does not affect overall consistency of the domain model, but is just concerned with a specific object being part of a particular operation.

(b) Removal of individual pre- and postconditions

An OCL operator consists of prevail conditions, necessary transitions and conditional transitions. A necessary transition describes a change in the state of an object in terms of a pre- and a postcondition within one structure. For example: for the operator *fetch_jack* the jack object changes state from being in the boot (precondition for the transition) to being available for use (postcondition for the transition).

```
% necessary
    [     sc(jack,J,[jack_in(J,C)]=>[have_jack(J)])],
```

Therefore it was not possible to delete individual pre- and postconditions because GIPO will only allow the insertion or deletion of whole transitions. It is impossible to remove by itself either a precondition or a postcondition - the removal of one automatically means the removal of the other. This means we remove the whole transition and so has the effect of removing one of the objects involved in an operator. We would not expect this type of error to be detected by validation checks - but it does become apparent when using the stepper.

4.3 Impossible Goal

The error introduced was of attempting to satisfy an impossible goal, that two hubs were simultaneously jacked up by the same jack object.

```
% Goals
[
     se(hub,hub0,[jacked_up(hub0,jack0),fastened(hub0)]),
     se(hub,hub1,[jacked_up(hub1,jack0),fastened(hub1)])]

% INIT States
    [
     ss(container,boot,[open(boot)]),
     ss(nuts,nuts_1,[loose(nuts_1,hub1)]),
     ss(nuts,nuts_0,[loose(nuts_0,hub0)]),
     ss(hub,hub0,[on_ground(hub0),fastened(hub0)]),
     ss(hub,hub1,[on_ground(hub1),fastened(hub1)]),
     ss(pump,pump0,[pump_in(pump0,boot)]),
     ss(wheel,wheel0,[wheel_on(wheel0,hub0)]),
     ss(wheel,wheel1,[wheel_on(wheel1,hub1)]),
     ss(wheel,wheel2,[have_wheel(wheel2)]),
     ss(wrench,wrench0,[have_wrench(wrench0)]),
     ss(jack,jack0,[have_jack(jack0)])]
```

The initial state is that the two hubs are on the ground, with wheels on, nuts loose, and the jack and wrench available.

Again the validation checks don't report an impossible task. When tested with the planners the simple forward planner found no solution. The FF planner very quickly says the problem is proven unsolvable. Using the stepper we can jack up the first hub - but then the jack isn't available for jacking up the second hub. The state of the jack is *jack_in_use*, which persists after jacking up the first hub.

Our conclusion is that an additional constraint is required in the model to say that a jack can't be in use on two different hubs. In addition, it would be useful to have an additional validation check within GIPO to detect this type of error.

5 Introducing Errors in Tyres World (B-AMN)

After errors were introduced, the new specification was validated using animation and proof, for these two activities are complementary [8]. The proof obligation generator checks that the initial state of the machine obeys the invariant, and that each operation preserves it. It also checks that the 'context' is consistent (the underlying sets, constants and their properties). For the 'correct' version of tyres world, 64 proof obligations were generated out of which 33 were automatically discharged. For the rest, it is necessary for some interaction with the prover. (Often the obligations can be seen to be true by inspection.)

However what we found particularly useful in these experiments is the fact that it is possible for the invariant to be at least partially evaluated and displayed during an animation, as well as the state values 'pre and post' to be displayed. This feature provided us with some surprises - as we discovered violations of the invariant where we expected none - showing the 'correct' B-AMN model to be in fact faulty. A useful feature of the BTool is that its source files are in ASCII and this is translated to LaTeXto provide the appropriate mathematical symbols. The initial values of the machine variables are shown as animation output (which

is also in ASCII). Variables `Tight`, `Loose` etc. are functions and the 'maplet' `|->` denotes a functional link.

```
Tight     {nuts1 |-> Hub1 , nuts2 |-> Hub1 , nuts3 |-> Hub3 ,
           nuts4 |-> Hub4}
Loose     {}
Hold_Nuts {nuts1 |-> FALSE , nuts2 |-> FALSE , nuts3 |-> FALSE,
          nuts4 |-> FALSE}
WheelHub  {Wl1 |-> Hub1 , Wl2 |-> Hub1 , Wl3 |-> Hub3 , Wl4 |-> Hub4}
Open      {boot |-> FALSE , tool_box |-> FALSE}
ToolIn    {wrench |-> boot , jack |-> boot , pump |-> boot}
HubUp     {}
WheelIn   {spare_wl |-> boot}
```

For a 'correct' initial state the invariant should be true. The invariant 'check' included the constraint that nuts cannot be both tight and loose - the domains of *Tight*, *Loose* must have an empty intersection. Initially (as above) this is true:

```
dom({nuts1 |-> Hub1 , nuts2 |-> Hub2 , nuts3 |-> Hub3 , nuts4 |->
      Hub4}) /\ dom({}) = {}
       true
```

5.1 Errors in Initialisation

Some deliberate errors were introduced to replicate as far as possible the deliberate errors introduced in OCL.

(a) 2 wheels and 2 nuts on one hub

The first initialisation error was to introduce two sets of wheels and nuts to one wheel. The invariant insists that the function *WheelHub* relating wheels to hubs and the functions *Tight*, *Loose* relating nuts to hubs should all be 1-1. These conditions are all contravened as can be seen in attempt to check the function *Tight*:

```
{nuts1 |-> Hub1 , nuts2 |-> Hub1 , nuts3 |-> Hub3 , nuts4 |-> Hub4} :
    {nuts1 , nuts2 , nuts3 , nuts4} >+> {Hub1 , Hub2 , Hub3 , Hub4}
```

where `>+>` is the ascii for an injective function. As can be seen this is *not* an injective function. However BTool does not attempt to evaluate it, but leaves it to be determined by the tool user.

The mistake is pinpointed by BTool when we subsequently attempt to Loosen nuts. The following conjunct of the invariant checks that nuts can be loose on a hub, or tight on a hub, but not both:

```
ran({nuts4 |-> Hub4 , nuts2 |-> Hub1 , nuts3 |-> Hub3})
  /\ ran({nuts1 |-> Hub1}) = {}
 false
```

This is clearly found to be false. The result of this experiment is that BTool finds this error through animation and the check of the corresponding invariants.

(b) Nuts on hub but no wheel
An initial state was attempted with nuts on Hub2 but no wheel on that hub.

```
  WheelHub  {Wl1 |-> Hub1 , Wl3 |-> Hub3 , Wl4 |-> Hub4}
   Tight    {nuts1 |-> Hub1 , nuts2 |-> Hub2 , nuts3 |-> Hub3 ,
             nuts4 |-> Hub4}
```

The invariant is checked before we start any other animation operations. The relevant conjunct is that if there are nuts on a hub (either tight or loose) then there is a wheel on that hub (<: means 'subset'):

```
   ran(Tight) \/ ran(Loose) <: ran(WheelHub)
/* which evaluates to */
   ran({nuts1 |-> Hub1 , nuts2 |-> Hub2 , nuts3 |-> Hub3 ,
                 nuts4 |-> Hub4}) \/  ran({})
   <: ran({Wl1 |-> Hub1 , Wl3 |-> Hub3 , Wl4 |-> Hub4})
       false
```

As can be seen, the result is a false invariant and this is picked up immediately by the tool.

5.2 Pre- and Postcondition Errors

(a) Removal of wrench condition:
The first error introduced was that the condition for a wrench being available was removed. The following shows

```
Current State

  Tight     {nuts4 |-> Hub4 , nuts1 |-> Hub1 , nuts2 |-> Hub2}
  Loose     {nuts3 |-> Hub3}
  Hold_Nuts {nuts1 |-> FALSE , nuts2 |-> FALSE , nuts3 |-> FALSE ,
             nuts4 |-> FALSE}
  WheelHub  {Wl1 |-> Hub1 , Wl2 |-> Hub2 , Wl3 |-> Hub3 , Wl4 |-> Hub4}
  Open      {boot |-> FALSE , tool_box |-> FALSE}
  ToolIn    {wrench |-> boot , jack |-> boot , pump |-> boot}
  HubUp     {}
  WheelIn   {spare_wl |-> boot}
```

In this case there was no problem with contravention of the precondition and no problem with invariant. The error is only demonstrated by the showing of a 'silly' result in that the wrench is still in the boot. This is a 'domain-specific' error, as it was for GIPO, which could only have been demonstrated by animation.

(b) Removal of precondition for putting away wheel
The error here was the removal of 'wheel not on hub' precondition so that any wheel could be put away[3]. In this case the operation took place with no problems in the *pre-operational*

[3]This was a non-deliberate error in the original B-AMN model.

invariant or precondition. However the invariant *post* the operation showed a 'false', in its evaluation of the predicate modelling the condition that a wheel cannot be both in the boot and on a hub:

```
dom({Wl2 |-> boot , Wl1 |-> boot}) /\ dom({Wl3 |-> Hub3 ,
   Wl4 |-> Hub4 ,  Wl2 |-> Hub2}) =  {}
     false
```

The result here is that this error is discovered by BTool when an attempt is made to invoke an operation which would result in a false invariant.

(c) PostCondition errors:
A postcondition was removed from the operation which replaced the nuts, that the nuts would be on the wheel and loose. The relevant invariant conjunct is that each nut is in a state 'tight or loose or held':

```
dom({nuts4 |-> FALSE , nuts1 |-> FALSE , nuts2 |-> FALSE , nuts3 |->
                   FALSE}  |> {TRUE}) \/
 dom({nuts3 |-> Hub3 , nuts1 |-> Hub1 , nuts2 |-> Hub2}) \/ dom({})
  =  {nuts1 , nuts2 , nuts3 , nuts4}
       false
```

As can be seen, BTool evaluates this as 'false': invoking this operation would result in a false invariant.

5.3 Impossible Goal

There is no equivalent in BTool in 'setting impossible goals'. B AMN in its *machine* construct has no equivalent to 'one operation following another'. The only manner in which a sequence of operations can be specified is if a *refinement* of the machine is created and this was not tried. In order to correspond with the GIPO task, a single operation, of attempting to jack up a wheel where the jack is already in use (on *Hub*4) was tried. *HubUp* is a partial function linking a hub with the (singleton) set of jack. Prior to the operation we have:

```
HubUp      {Hub4 |-> jack}
```

After loosening nuts on another hub (*Hub*3) an attempt was made to jack this up. The precondition for the use of a jack is that it should not be already linked to a hub, This is shown, where `/:` means 'not a member of':

```
jack /: ran HubUp
 is false
```

BTool evaluated this as false since the (only) jack was already in use.

6 Conclusions and Further Work

Our tests have shown that neither the validation checks within GIPO nor the proof assistant within the BTool are sufficient on their own to detect the types of errors we introduced. It is necessary to use the stepper in GIPO and the animator in BTool to detect certain types of error and inconsistency.

The experiments also uncovered a previously unknown omission in each of the two domain models. In the case of the OCL this involved the missing 'inconsistency constraint' in the use of the jack (see Section 4.3). In the case of the B-AMN this was a missing precondition for putting away the tyre (see Section 5.2b).

It would seem that useful additional validation checks within GIPO would be (i) a tool to test the consistency of the initial state - it is a waste of time trying to generate a plan when the initial state is incorrect; (ii) a tool to test the consistency of the goal - again it is better to detect an error of this type when the task is created. These additional checks would further exploit the structure and knowledge of an OCL specification.

Work presented in this paper has also shown some correspondence between structures in OCL and B-AMN. Further investigation is required to confirm whether there is potential for an automated translation tool from one language to the other in order to exploit the powerful validation capabilities of the BTool.

References

[1] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[2] T. J. Grant. Topwards a Taxonomy of Erroneous Planning. In *Proceedings of the 20th UK Planning and Scheduling SIG, Edinburgh*, 2001.

[3] J. Hoffmann. A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-climbing Algorithm. In *Proceedings of the 14th Workshop on Planning and Configuration - New Results in Planning, Scheduling and Design*, 2000.

[4] B-Core (UK) Ltd. http://www.b-core.com/.

[5] T. L. McCluskey. The OCL Ontology. Technical report, Department of Computing and Mathematical Sciences, University of Huddersfield , 2001.

[6] T. L. McCluskey, D. Liu, and R. M. Simpson. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In *The Thirteenth International Conference on Automated Planning and Scheduling*, 2003.

[7] T. L. McCluskey and J. M. Porteous. Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency. *Artificial Intelligence*, 95:1–65, 1997.

[8] P. Mukherjee. Computer-aided validation of formal specifications. *Software Engineering Journal*, 10(4):133–140, July 1995.

[9] J. Penix, C. Pecheur, and K. Havelund. Using Model Checking to Validate AI Planner Domain Models. In *Proceedings of the 23rd Annual Software Engineering Workshop, NASA Goddard*, 1998.

[10] J. Rushby. Formal Methods and the Certification of Critical Systems. Technical Report CSL-93-7, Computer Science Laboratory, SRI International, Menlo Park CA 94025 USA, December 1993.

[11] S. Russell. Efficient memory-bounded search algorithms. In *Proc. ECAI*, 1992.

[12] S. Schneider. *The B-Method: An Introduction.* Palgrave, 2001.

[13] R. M. Simpson, T. L. McCluskey, W. Zhao, R. S. Aylett, and C. Doniat. GIPO: An Integrated Graphical Tool to support Knowledge Engineering in AI Planning. In *Proceedings of the 6th European Conference on Planning*, 2001.

[14] M. M. West, D. E. Kitchin, and T. L. McCluskey. Validating Planning Domain Models Using B-AMN. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling SIG, Delft, Netherlands*, 2002.

[15] M. M. West and T. L. McCluskey. The application of machine learning tools to the validation of an air traffic control domain theory. *International Journal on Artificial Intelligence Tools*, 10(4):613 – 637, December 2001.