

Search guidance with composite actions

- Increasing the understandability of the domain model

Vägledning med sammansatta handlingar

- Förbättring av förståbarheten i domänmodellen

Erik Hansson

Tutor: Mikael Nilsson

Examiner: Jonas Kvarnström

Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>.

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>.

Abstract

This report presents an extension to the domain definition language for Threaded Forward-chaining Partial Order Planner (TFPOP) that can be used to increase the understandability of domain models. The extension consists of composite actions which is a method for expressing abstract actions as procedures of primitive actions. TFPOP can then use these abstract actions when searching for a plan. An experiment, with students as participants, was used to show that using composite action can increase the understandability for non-expert users. Moreover, it was also proved the planner can utilize the composite action to significantly decrease the search time. Furthermore, indications were found that using composite actions is equally fast in terms of search time as using existing equivalent methods to decrease the search time.

Acknowledgments

To begin with I would like to express my gratitude to Jonas Kvarnström and Mikael Nilsson for the feedback and help during the work. Moreover, Mikael also has my gratitude for all the discussions about planning which was only remotely connected to this thesis. They definitely proved to be interesting.

I would also like to give my thanks to Örjan Dalhström who gave me some of his time to answer my questions regarding statistics, even if he had no obligation whatsoever to do so. Furthermore, Ola Leifler deserves my thanks for putting up with my questions all the times I dropped by his office unannounced.

Finally, to my family, friends and colleagues who have helped me, knowingly or unknowingly, by dropping by for a coffee break, guilt tripping me to leave the office in the evenings or simply talk about something else. All the small things made wonders for the productivity and therefore, you have my deepest thanks.

Linköping, June 2016
Erik Hansson

Contents

| | |
|---|-------------|
| Abstract | iii |
| Acknowledgments | v |
| Contents | vi |
| List of Figures | viii |
| List of Tables | ix |
| 1 Introduction | 1 |
| 1.1 Motivation | 1 |
| 1.2 Aim | 2 |
| 1.3 Research questions | 2 |
| 1.4 Delimitations | 2 |
| 2 Background | 5 |
| 2.1 Planning | 5 |
| 2.2 Threaded Forward-chaining Partial Order Planner | 8 |
| 3 Theory | 17 |
| 3.1 Definitions and notations | 17 |
| 3.2 Composite actions | 17 |
| 3.3 Understandability measurement | 20 |
| 4 Method | 25 |
| 4.1 Composite action component ranking | 25 |
| 4.2 Implementation | 25 |
| 4.3 Search time | 26 |
| 4.4 Understandability measurement | 27 |
| 4.5 Understandability experiment | 28 |
| 5 Results | 33 |
| 5.1 Composite action component ranking | 33 |
| 5.2 Implementation | 34 |
| 5.3 Search time | 43 |
| 5.4 Understandability measurement | 45 |
| 5.5 Understandability experiment | 45 |
| 6 Discussion | 47 |
| 6.1 Results | 47 |
| 6.2 Method | 50 |
| 6.3 Source criticism | 51 |
| 6.4 The work in a wider context | 52 |

| | |
|---|-----------|
| 7 Conclusion | 53 |
| 7.1 Research questions | 53 |
| 7.2 Further work | 53 |
| A Metric formulas | 55 |
| A.1 Code and data spatial complexity | 55 |
| A.2 Improved cognitive information complexity | 55 |
| B Blocks World | 57 |
| C Questionnaire | 63 |
| D PDDL Domain Introduction | 77 |
| E Search Time - Compared to no Guidance | 81 |
| F Data understandability pre-experiment | 83 |
| G Data understandability experiment | 85 |
| Bibliography | 87 |

List of Figures

| | | |
|------|---|----|
| 2.1 | A causal link example | 7 |
| 2.2 | TFPOP plan representation showing the ordering rules | 9 |
| 2.3 | TFPOP search structure when selecting a primitive action | 10 |
| 2.4 | TFPOP search structure when selecting a sequence action | 11 |
| 2.5 | The people-in-distress domain: flags, types, constants and fluents | 12 |
| 2.6 | The people-in-distress domain: operators part 1 | 13 |
| 2.7 | The people-in-distress domain: operators part 2 | 14 |
| 2.8 | The people-in-distress domain: sequence operators | 15 |
| 2.9 | The distressed people at mountain problem instance | 16 |
| | | |
| 4.1 | Method for searching for understandability measurements | 28 |
| | | |
| 5.1 | The syntax for composite actions in TDDL | 35 |
| 5.2 | Composite action branch point that adds to the plan | 36 |
| 5.3 | Composite action branch point that does not add to the plan | 36 |
| 5.4 | The syntax for variable introduction in TDDL | 39 |
| 5.5 | Branches of a with where branch point | 40 |
| 5.6 | The syntax for an if statement in TDDL | 40 |
| 5.7 | Branches of an if branch point | 41 |
| 5.8 | Evaluation of an if statement | 41 |
| 5.9 | The syntax for a while statement in TDDL | 42 |
| 5.10 | Branches of a while branch point | 42 |
| 5.11 | Evaluation of a while statement | 42 |
| 5.12 | The syntax for a sequence statement in TDDL | 43 |
| 5.13 | Evaluation of a sequence statement | 43 |
| 5.14 | Evaluation of a composite action | 44 |
| 5.15 | Search time, composite actions and equal guidance | 44 |
| | | |
| B.1 | Blocks world domain without any extra guidance. | 57 |
| B.2 | Blocks world domain with composite action as extra guidance. | 58 |
| B.3 | Blocks world domain with guidance equal to composite actions, part 1. | 59 |
| B.4 | Blocks world domain with guidance equal to composite actions, part 2. | 60 |
| B.5 | Blocks world problem used to compare search time when comparing to no guidance. | 60 |
| B.6 | Example randomized problem for composite action | 61 |
| B.7 | Example randomized problem for equivalent guidance | 61 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | The two common search spaces within planning | 6 |
| 3.1 | Notations used in the report | 18 |
| 3.2 | Software understandability aspects | 23 |
| 4.1 | All the search keywords sorted after categories | 28 |
| 5.1 | The ranking of the composite action components | 33 |
| 5.2 | The categorization of complex actions in Golog | 34 |
| 5.3 | The categorization of components in BFM | 34 |
| 5.4 | The categorization of components in the extension of TAL with composite actions | 34 |
| 5.5 | The categorization of components in HTN | 34 |
| 5.6 | Composite action component to implementation map | 37 |
| 5.7 | Label suffix for memory objects | 38 |
| 5.8 | Search time, composite actions and no guidance | 44 |
| A.1 | The cost for different basic control structures according to Wang and Shao. | 56 |
| E.1 | Run time for domain with composite action | 81 |
| E.2 | Run time for domain without any search guidance | 82 |
| F.1 | Understandability pre-experiment data | 83 |
| G.1 | Understandability experiment data | 85 |



1 Introduction

1.1 Motivation

There are crises that result in situations that are not healthy for a human to be in. For example, it can be high radiation levels in the case of a reactor leak due to an earthquake. Naturally, this makes it harder to repair any damages. However, robots would be able to enter the reactor and make repairs without risking radiation sickness. Moreover, imagine that repair robots work in collaboration with unmanned aerial vehicles (henceforth, called UAV) to solve the problem. The UAV could be used to deliver the needed parts and supplies for the repair. This would make it possible to solve the whole crisis without any humans getting near the risk zone.

Before the example in the previous section can become reality, there are challenges that need to be overcome. One of them is that the robots need to have a plan for how to find everything that has been damaged and repair it. Otherwise, they might wander around randomly until everything is repaired. Two main components are required to create such a plan: A planner (a program that searches for a plan given a model of the real world called a domain model) and a domain model.

A common way to model a domain is to use the Planning Domain Definition Language (PDDL). A drawback with this, according to Strobel and Kirsch, is that it becomes hard to understand when the model grows in size [19]. Therefore, it also becomes harder to create and maintain. Moreover, this becomes an even harder challenge when the planner is a domain-configurable planner since a domain-configurable planner requires extra information about the domain (called domain knowledge) to guide the planner when it creates a plan [16].

There exist tools designed to make it easier for the user to understand a domain model written in PDDL. For example, Strobel and Kirsch developed an integrated development environment (IDE) [19]. However, these are all tools that in some way provide functionality to work with PDDL (see Shah et al. for an overview of the existing tools as of 2013 [18]). In this report another approach is taken to increase the understandability of the domain model. Instead of providing the user with more tools, an extension to the language for modeling used by Threaded Forward-chaining Partial Order Planner by Kvarnström [14] is proposed. Henceforth, modeling language will be called TDDL (for Threaded planning Domain Definition Language) and the planner will be called TFPOP. The extension allows for expressing certain types of domain knowledge in a less complex manner through allowing composite

actions to be defined in the domain in addition to the already supported primitive actions. Composite actions can be used to express simple sequences of actions (for example, pick up item a then go to B and deliver item a) or for more complex action sequences (for example, while there are items at A , first go to A and pick up an item x , then go to B and deliver x). The planner can use these composite actions when creating a plan, instead of selecting all the individual components one by one.

As stated in the previous section, the proposed extension is aiming to be less complex than expressing the domain knowledge without it. Therefore, the lower complexity would make the model more understandable according to Bansiya and Davis since they identified complexity as one of the factors that affect understandability negatively [3].

In addition to presenting the extension, this report aims to show that composite actions can be used to increase the understandability. Moreover, it will also prove that they can be implemented by implementing support for a subset of all the potential composite actions in TFPOP. Finally, the report will research whether a planner can utilize the extra domain knowledge by testing the performance when using them in TFPOP.

1.2 Aim

The main goal of this report is to create an extension to TDDL that allows expressing some domain knowledge in a more understandable way. Moreover, it also aims to verify that the extension is more understandable and that it is possible to utilize the domain knowledge to decrease the search time.

1.3 Research questions

There are three main research questions in this report. Dividing these three into smaller sub-questions and identifying questions that need to be answered before the main question results in the following research questions:

1. Can composite action components (for example, sequence and while loops) be implemented in TFPOP?
 - a) Which composite action components should be implemented?
2. How well can TFPOP utilize the domain knowledge in the composite actions when searching for a plan?
 - a) Can extra domain knowledge through composite actions be used to decrease search time?
 - b) Is there an indication that extra domain knowledge through composite actions is at least as good as equivalent domain knowledge without composite actions in terms of search time to find a plan?
3. Can composite actions increase the understandability of a domain model in TDDL compared to using the equivalent guidance without composite actions?
 - a) How can understandability be measured for a domain model written in a TDDL?

1.4 Delimitations

In the report there are three delimitations. Firstly, the report will not try to prove that all composite action components can be implemented in TFPOP (question 1) due to the time constraints of this report. Moreover, only a subset of all possible composite action components will be considered when deciding which composite action components that should be implemented in TFPOP (question 1a). This restriction is applied since the main goal of

this report is not to decide which composite action components that are most important but whether understandability can be increased by introducing composite actions.

Secondly, the report will not attempt to prove or disprove any difference of the effect of the composite actions and the effect of the equivalent domain knowledge without composite actions in terms of search time (question 2b). Instead, only a small test will be conducted to get an indication regarding the previously mentioned equivalence.

Finally, the report will not explore if composite actions always increase understandability (question 3). This is because composite actions only express one type of domain knowledge. Therefore, they cannot be used to increase the understandability in the general case.



2 Background

The focus of the background chapter is to present the relevant information about TFPOP and TDDL that is needed to understand how composite actions can be implemented in the planner. However, this requires some basic knowledge about planning and the types of planners that TFPOP is based on. Therefore, the first section will cover this. The following section will cover parts about TFPOP and TDDL relevant for this report.

2.1 Planning

One important part in planning is to search for a plan that solves the given problem [17]. The problems (in classical planning as described by Russell and Norvig [17]) can be formulated as: Given a certain situation (called the initial state), find an unordered, partially ordered or totally ordered set of actions (called a plan) that changes the initial state to another situation (called a goal state). For example, a problem within planning could be (this example will be a recurring example through the background chapter): Given a UAV at the base, a crate with supplies at the base and some people in distress on an isolated mountain, find a set of actions that ensures that the people in distress has the crate with supplies and that the UAV has returned to the base. To solve this problem a planner has to search through all the possible plans for a valid plan (one that changes the initial state to a goal state).

There are other ways of formulating a planning problem used in non-classical planning. For example, Hierarchical Task Networks formulates the planning problem as a high-level action that needs to be refined into a list of high-level action and primitive actions [17]. This continues until there is only a sequence of primitive actions left. However, this is still a search problem since there can be multiple ways to refine a task. All the same, the other ways of formulating a planning problem will not be covered further in the background chapter since it is not needed to understand TFPOP.

If a planner is to search for a plan that solves a problem then it needs information about what can be done to change the situation (called the current state). For example, solving the problem mentioned above is in itself impossible since the planner does not know what can be done. However, if the planner gets the extra knowledge that a UAV can perform the following actions, then it has enough information:

- **Start:** The UAV starts flying.

- **Pick up:** The UAV picks up a crate. This requires that the crate is at the same location as the UAV and that the UAV is not holding another crate.
- **Move:** The UAV moves to a new location.
- **Deliver:** The UAV delivers the crate it is holding to its current location. This requires that the UAV is holding a crate.
- **Land:** The UAV lands at its current position.

When an action is defined in classical planning it has properties that describe when it can be used, called preconditions, and what it does, called effects. The preconditions state what needs to be true for an action to be applicable. For example, **pick up** has the preconditions that there need to be a crate at the same place as the UAV and it may not already hold a crate. The effects are the changes that the action make to the current state. In the example with **pick up**, the effect is that the UAV is holding the crate and that the crate is no longer at the place it was.

The actions together with the types of objects that can exist, the properties and relations that objects can have and all constant objects are called the domain. This is different from a problem instance which exists in a domain and describes which objects exist, an initial state and a goal state.

Given both the domain and the problem instance a planner is able to search for a plan that guarantees that the goal is reached. Exactly how the search is conducted depends on the planner. However, the search is usually conducted using a search space (a graph) that is a state space or a plan space [17]. The different search spaces are explained in table 2.1.

| Graph representation | State space | Plan space |
|----------------------|--|---|
| Node | A state | A plan |
| Directed edge | An action | An operator that changes the plan (for example, add or remove an action) |
| Start node | The initial state | An empty plan |
| A goal node | A state that upholds all the conditions in the goal state | A plan that changes the initial state to a goal state |
| A path | A plan changing the state at the first node in the path to the state at the end node in the path | A sequence of operators that modifies the plan in the first node of the path to the plan in the last node of the path |

Table 2.1: The two common search spaces within planning.

Forward-chaining planning

Forward-chaining planning (also called forward or progression state-space search) is a category of planners that searches through a state space from the initial state to a goal state. The basics are quite simple for this kind of planners. Namely, select a node in the search tree and one action that is possible to apply in the corresponding state that has not been tried before. Thereafter, create a new child node whose state is computed by applying the effects of the selected action to the state of the selected node. Finally, check if the new state fulfills the goal and if it does not repeat the procedure for a new state and/or action until the goal is reached.

The drawback with this kind of planning is that there can be a huge branching factor in each choice. For example, TFPOP was tested with problem instances that could contain up to 512 crates, 16 UAVs, 8 ground robots, 16 carriers for the UAV and more in an article by Kvarnström [14]. Lets assume that all the UAVs are at different locations and each location

has 16 crates. Furthermore, if there is one action that makes a UAV move a crate to its destined location and returns the UAV to the location it came from, then there are 512 different combinations of variables for that action. This is among the best case scenarios where the worst being where all crates and UAVs are at the same location (8192 possible combinations). Naturally, there are scenarios where there are fewer options. However, no other action is included in this number. Hence, it is safe to assume that 512 is the lowest average branching factor for the problem instance. Finally, lets assume that the problem instance has the goal that 16 crates has to be moved to the correct location and all UAVs are at the same place they started at. Using these assumptions, there are 512^{16} ($\approx 2.23 * 10^{43}$) possible combinations of actions with variable bindings in total. Even if a processor could test one combination per cycle and thread, it would take approximately $1.18 * 10^{25}$ years to check all the combinations¹. Obviously, this is too long time.

To handle this complexity a forward-chaining planner uses methods to guide the search. One example is to use a search heuristic that estimates the distance to the goal through the number of conditions in the goal state that is fulfilled [17]. In the people in distress example this would mean that **deliver** is tried before **move** when an UAV is at the isolated mountain with the filled create since it results in that the people in distress gets their supplies.

Partial order planning

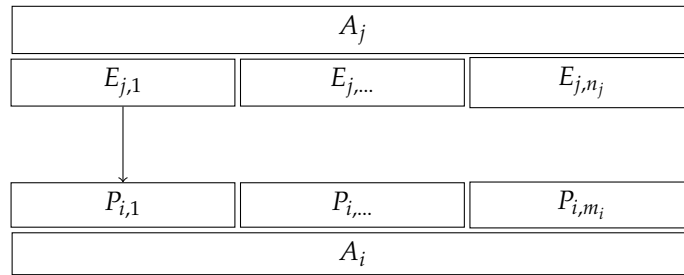


Figure 2.1: A causal link from the effect $E_{j,1}$ in action A_j ensuring that the precondition $P_{i,1}$ in action A_i holds. n_j is the number of effects of action A_j and m_i is the number of preconditions of action A_i

Partial order planning is a type of planning that, unlike forward-chaining planning, searches through a plan space. A plan in partial order planning consists of a partially ordered set of actions [21]. The following paragraphs describe the partial order causal link (henceforth, called POCL) approach for creating partially ordered plans.

In POCL a plan consists of a set of actions, a set of ordering rules for the actions and a set of so called causal links [21]. A causal link is a link going to precondition m ($P_{i,m}$) of an action (A_i) from effect n ($E_{j,n} == P_{i,m}$) of another action (A_j), as shown in figure 2.1. The causal link models that the effect of A_j ensures that the precondition in A_i is true and is written as $A_j \xrightarrow{P_{i,m}} A_i$. These causal links can show threats to the plan if another action (A_k) with the effect $E_{k,x} == \neg P_{i,m}$ is added to the plan. For example, if A_j is executed and thereafter A_k then $\neg P_{i,m}$ will be true and therefore A_i can not be executed. Hence, adding A_k to the plan also adds a threat to the plan. To resolve this threat, the planner adds an ordering that says that A_k has to be executed before A_j or after A_i . Following below is an example of a POCL algorithm presented by Weld [21].

At the start the algorithm, it creates a starting plan represented by a triple consisting of actions, ordering rules and causal links ($\langle A, O, L \rangle$) and an agenda where:

¹The Intel Core i7-5960X can test approximately $6 * 10^{10}$ combinations per cycle (using data from www.intel.se accessed 30th of May 2016).

- The actions in the starting plan is an initial action H_0 and a goal action H_∞ ($A = \{H_0, H_\infty\}$) where:
 - H_0 is an action with no preconditions and its effect is equal to the initial state.
 - H_∞ is an action with no effect and its preconditions are equal to the conditions in the goal state.
- The ordering rules is a set consisting of the rule $H_0 < H_\infty$ that states that H_0 has to happen before H_∞ ($O = \{H_0 < H_\infty\}$).
- The causal links are an empty set ($L = \{\}$).
- The agenda consists of all the preconditions in H_∞ ($agenda = \{precond(H_\infty)\}$, where $precond$ is a function that extracts the preconditions of the action that is its argument).

The algorithm then iterates over the following steps until *agenda* is empty:

1. Chose a precondition $P_{i,m}$ for action A_i from *agenda*.
2. If there exist an action A_j in A with the effect $E_{j,n} == P_{i,m}$, add $A_j \xrightarrow{P_{i,m}} A_i$ to L and $A_j < A_i$ to O .
3. Else, choose an action A_j from all the possible actions in the domain that has an effect $E_{j,n} == P_{i,m}$. Add the action A_j to A and add $A_j \xrightarrow{P_{i,m}} A_i$ to L . Furthermore, add $A_j < A_i$, $A_0 < A_j$ and $A_j < A_\infty$ to O and add $precond(A_j)$ to *agenda*.
4. Add new ordering rules to O that resolve all unresolved threats.
5. Remove $P_{i,m}$ from *agenda*.

When the algorithm terminates it has found a plan consisting of partially ordered actions which all have their preconditions satisfied by either the initial action or by another action. Moreover, the resulting plan contains no threats. Though most importantly, the plan solves the planning problem.

2.2 Threaded Forward-chaining Partial Order Planner

Threaded Forward-chaining Partial Order Planner is a planner that uses ideas from two different types of planning. The first type is forward-chaining planning and the second is partial order planning [14]. In addition to this, TFPOP is built to create plans for multiple agents executing actions in parallel. When searching for a plan, each agent has a thread to which actions can be added. An important part is that all the actions in one thread are totally ordered. In contrast, actions in different threads are only partially ordered. Moreover, each thread keeps track of what it knows as its current state. This enables TFPOP to make use of the knowledge in the current state, just like in forward-chaining planning [14].

The cost for the partial order between the threads is that the information in a state is not always complete since an action in another thread might affect the information. As a result, a variable in a state is not necessarily known to have one specific value but can have multiple possible values [14]. However, it is important to note that "state" is used to refer to what can be inferred to be true in a thread (the states that TFPOP stores) and not what is true (the state of the world). If required, TFPOP can add a causal link between actions in different threads to ensure that a variable has exactly one of the possible values [14].

In TFPOP there is an extra type of actions in addition to the primitive actions. This type is a sequence of primitive actions called sequence action. TFPOP can use these to add a sequence of primitive actions to the plan instead of the one action it adds when using a primitive action. For example, it is quite obvious for the one that models the people-in-distress domain that **pick-up** crate with supplies followed by **move** to isolated mountain, then **deliver** supply crate and then **move** to base is a sequence of actions that needs to be done to deliver a crate and return for delivering more or landing. By modeling this sequence of primitive actions as a sequence action, TFPOP only has to find the sequence action instead of all four primitive

actions (in the correct order and with the correct arguments) to include the delivery of a crate in the plan.

The following sections will cover the parts of TFPOP that are essential for implementing composite actions. Following that is a description of how a domain and a problem instance is written in TDDL. All the information in the following sections comes from the source code of TFPOP.

Plan structure

A plan in TFPOP is represented by three main components: Nodes that represent the start and the end of actions (called invocation and effect nodes, respectively), causal links between these nodes and a set of ordering rules. Causal links and ordering between the nodes in the plan work just like in POCL planning except that they go from an effect node to an invocation node instead of from one action to another.

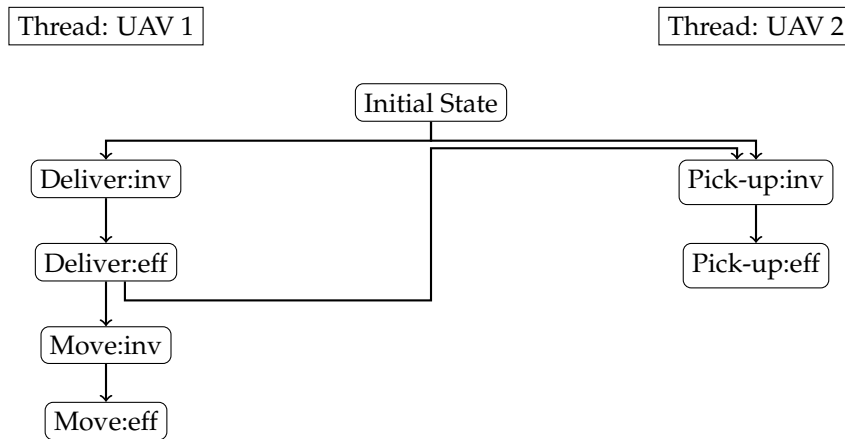


Figure 2.2: A TFPOP plan representation in the people-in-distress domain showing ordering rules as arrows. UAV 1 is holding a crate in the initial state. Furthermore, UAV 1 and UAV 2 are at the same location in the initial state.

An example plan in the people-in-distress domain is presented in the figure 2.2. The graph shows that all the nodes in a thread are totally ordered. Moreover, there is an arrow going between the UAV 1 thread to the UAV 2 thread (from node *Deliver:eff* to invocation node *Pick-up:inv*). This ordering rule is necessary since there is a causal link, which is not shown in the graph, going from the effect node *Deliver:eff* to invocation node *Pick-up:inv* that ensures that there is a crate for UAV 2 to pick up. Without this ordering rule there would be nothing that states that *Pick-up:inv* has to happen after *Deliver:eff*. Hence, when executed there is a risk the action **Pick-up** used by UAV 2 would fail due to the fact that UAV 1 is currently holding the crate that UAV 2 should pick up.

Searching

The search that TFPOP does to find a plan is usually a standard depth first search. However, it is guided with extra domain knowledge that is supplied through the domain specification. During the search, TFPOP visits different kinds of search nodes. These are called branch points and their children are called branches. The following types of branch points exist at the moment:

- **Thread branch point:** TFPOP chooses which thread, and by extension agent, that it should add an action to. By default, TFPOP makes this choice based on the expected

time it takes for the threads to execute all their actions under the assumption that they are applied without delays. The first choice is the one which is expected to finish first and the last is the one that is expected to finish last.

- **Arbitrary action branch point:** TFPOP chooses which action to add to the plan given a thread and agent. To make this choice all actions are divided into three sets: The actions that are known to be applicable in the current state; those that are known to not be applicable; and those that might be applicable. TFPOP starts with nondeterministically selecting one from the set that is known to be applicable. If that set is empty, then TFPOP continues with the set that might be applicable. In addition to selecting which action to add, this branch point also determines how to make the preconditions of the selected action true. This is done by selecting values for the variables and adding causal links from effect nodes already in the plan to the invocation node associated with the selected action that ensure its preconditions holds.
- **Sequence action branch point:** This is similar to the arbitrary action branch point except that it only has one possible primitive action to choose from and the variables are already selected. That leaves for the branch point to select which causal links that should be added to ensure that all the preconditions are guaranteed to be true. One extra part applies to the first primitive action in the sequence. Namely, that the causal links that ensure that the preconditions for the sequence action hold are connected to its invocation node.
- **Threat fixing branch point:** The planner choose how to solve any threats in the plan by adding ordering rules.
- **Simple action applier branch point:** TFPOP adds the invocation and effect nodes associated with the selected action to the plan. Moreover, if the action is marked as an action that can achieve the goal then the branch point will also check if the goal state has been reached.

When TFPOP visits a branch point, it might be the case that there is no possible branch left to choose. If that happens, TFPOP will backtrack to the parent node and mark that branch as tested. Thereafter, it will select a new branch if one exists and else backtrack further.

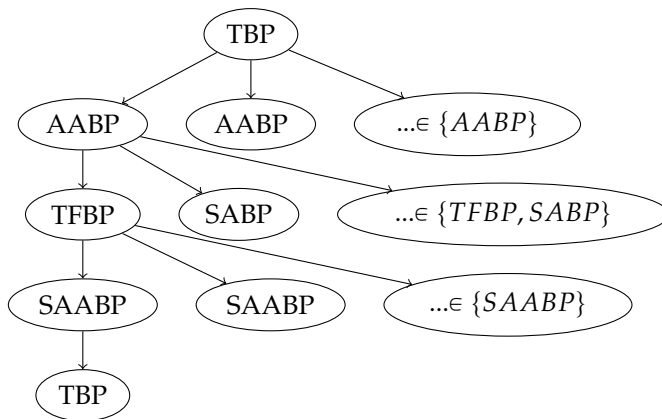


Figure 2.3: A part of the search tree that TFPOP searches through when selecting a primitive action. Each node is a branch point and each edge is a branch. The meaning of the abbreviations are: TBP is thread branch point; AABP is arbitrary action branch point; SABP is sequence action branch point; SAABP is a simple action applier branch point; and TFBP is threat fixer branch point.

The search that TFPOP does always start with a **thread branch point**. This branch point result in that the next branch point is an **arbitrary action branch point** unless there are no branches left. Assuming that the **arbitrary action branch point** chose a primitive action (sequence actions is covered in the next section), the next branch point is a **threat fixer branch point**. The next step after all threats are resolved is a **simple action applier branch point**. Finally, when the action has been applied, TFPOP is faced with a new **thread branch point**. A part of a search tree describing this sequence of choices is shown in figure 2.3.

If TFPOP selects a sequence action in an **arbitrary action branch point**, then all the primitive actions in the sequence will be applied. This is done through the process of cycling

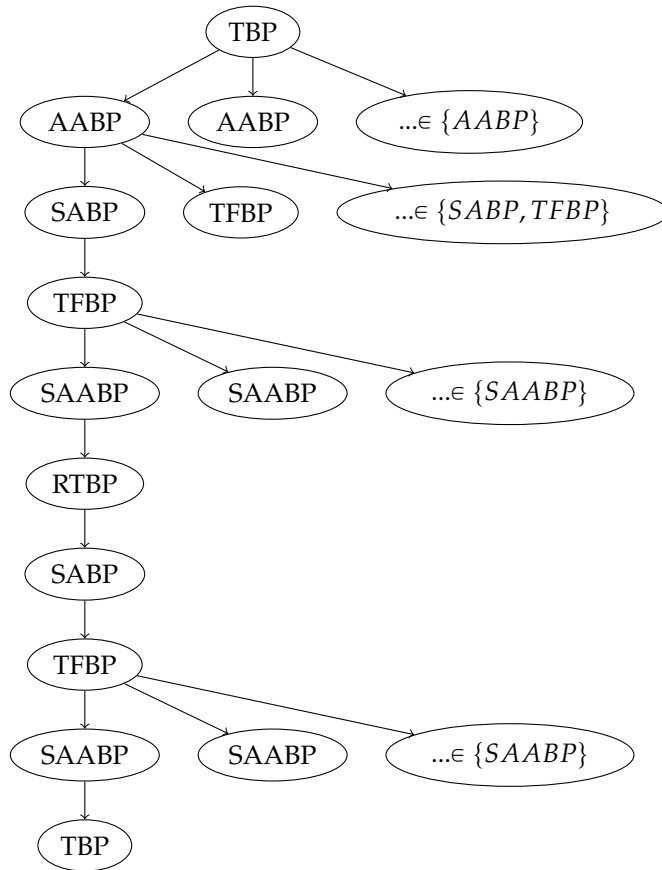


Figure 2.4: A part of the search tree that TFPOP searches through when selecting a sequence action. In this graph the arbitrary action branch point selected a sequence action of length two. Each node is a branch point and each edge is a branch. The meaning of the abbreviations are: TBP is thread branch point; RTBP is a thread branch point which is restricted to the last chosen thread; AABP is arbitrary action branch point; SABP is sequence action branch point; SAABP is a simple action applier branch point; and TFBP is threat fixer branch point.

through the following branch points until all the actions has been applied: **Sequence action branch point** followed by **threat fixing branch point**, **simple action applier branch point** and finally, **thread branch point**. In this cycle, all **thread branch points** but the last is restricted to choosing the thread that was chosen for the sequence action. When all primitive actions has been applied TFPOP will continue the search as usual. A part of a search tree describing this for a sequence action of length two is shown in figure 2.4.

To be able to keep track of where TFPOP is in a sequence action, all **sequence action branch point** has a state connected to them. These states contain information that is used by TFPOP to query if time constraints should be tested in the branch point or if it should test if a goal has been reached in the branch point. In the sequence action state, time constraints will only be tested at the end of the sequence and all goal checks are handled by the **simple action applier branch points**. In addition, the state is used to determine the next **sequence action branch point**, if one exits, in the sequence action.

The states mentioned above are not exclusive for the sequence action. **Arbitrary action branch points** also have a state associated with them. However, these states are simple since they only keep track of one action.

Modeling language

Modeling domains and problem instances in TFPOP is done with TDDL. TDDL is similar to PDDL when it comes to syntax since both of them have their base in LISP. Moreover, both languages use similar constructs in the language, even though there are some differences. As in PDDL, the domain is defined separately in TDDL, making it possible to reuse the domain for multiple problems. The following sections will cover how a domain and a problem instance is specified.

Domain specification

A domain defines which types of objects that can exist, which fluents exist, which constant objects exist and which actions can be used. The previously used people in distress example will be used to show how a domain is modeled in TDDL. The whole model is written in one file. However, it has been divided into three code snippets in this report. The first snippet of the domain model is shown in figure 2.5. This covers everything except the actions. The second snippet covers the actions (which are called operators in TDDL) and is shown in figures 2.6 and 2.7. Finally, the third snippet covers how a sequence action, see figure 2.8, is specified in TDDL.

```

1  (:tfpop-domain people-in-distress
2    (:flags)
3    (:types
4      (movables
5        (crate)
6        (people)
7        (agent
8          (uav)
9        )
10     )
11     (location
12       (base)
13       (mountain)
14     )
15   )
16   ;; The null-crate symbolizes that there is no crate
17   ;; (for example when not holding any crates)
18   ;; The same principle applies for null-loc but for locations
19   (:constants (crate null-crate) (location null-loc))
20   (:fluents
21     (location (at movables))
22     (boolean (is-empty crate))
23     (crate (holding agent))
24     (boolean (is-flying uav))
25   )

```

Figure 2.5: The people-in-distress domain: flags, types, constants and fluents.

The first part of the domain specification covers everything that is not an action. Everything that is included is shown in figure 2.5 and has the following meaning:

- "(:tfpop-domain people-in-distress)": This part specifies that the domain model is a domain for the TFPOP planner and that it is named "people-in-distress".
- "(:flags)": This part contains the flags that apply to the whole domain. However, they are not important for this report and will therefore be left out.
- "(:types": A specification of the types of objects that exist in the domain. The specification allows hierarchical types. For example, there are three different sub types of "movables" in this model: "crate", "people" and "agent".
- "(:constants": This part specifies all the objects that exist for all the problem instances that use the domain. In this case, there are two constants, "null-crate" and "null-loc" which model a non-existing crate respectively location. This can, for example, be used to describe that a crate held by an UAV is not at a location.
- "(:fluents": Every relation that exist in the domain is specified in this section of the domain. A relation can be between multiple objects, for example "(location (at movables))" states that a "movable" is at "location", or a property for a single object, for example "(boolean (is-flying uav))" is a property that states whether a "uav" is flying or not. Note that no values for the fluents are specified in this part, only that they exist. Finally, resources are also specified under fluents. However, they are not needed for this report they will be left out.

```

27 (:operator (start (uav ?uav) (thread ?thread))
28   (:split-precond
29     ?thread => (= (is-flying ?uav) false)
30   )
31   (:phase
32     (:duration 2 5 7)
33     (:effects
34       (:= (is-flying ?uav) true)
35     )
36   )
37   (:always-followed-by)
38   (:definitely-changes (is-flying ?uav))
39 )
40
41 (:operator :onlypart (pick-up (uav ?uav) (thread ?thread) (location ?loc)
42   (crate ?crate))
43   (:split-precond
44     ?thread => (and (= (holding ?uav) null-crate)
45                   (= (is-flying ?uav) true))
46     ?loc => (= (at ?uav) ?loc)
47     ?crate => (and (not (= ?crate null-crate)) (= (at ?crate) ?loc))
48   )
49   (:phase
50     (:duration 1 2 3)
51     (:effects
52       (:= (holding ?uav) ?crate)
53       (:= (at ?crate) null-loc)
54     )
55   )
56   (:always-followed-by)
57   (:definitely-changes (holding ?uav) (at ?crate))
58 )
59
60 (:operator :onlypart (move (uav ?uav) (thread ?thread) (location ?new-loc))
61   (:split-precond
62     ?thread => (= (is-flying ?uav) true)
63     ?new-loc => (and (not (= ?new-loc null-loc))
64                    (not (= (at ?uav) ?new-loc)))
65   )
66   (:phase
67     (:duration 10 20 40)
68     (:effects
69       (:= (at ?uav) ?new-loc)
70     )
71   )
72   (:always-followed-by)
73   (:definitely-changes (at ?uav))
74 )

```

Figure 2.6: The people-in-distress domain: operators part 1.

The second part of the domain model (shown in figures 2.6 and 2.7) describes all the actions that can be used within the domain. Each action consists of the following:

- Naming, onlypart flag and parameters: The first part of the action, directly after the keyword ":operator", is an optional flag (":onlypart") that states that the operator may only be used as part of a sequence action. Following that is a list where the first item is the name of the action, the second is the agent parameter and the third is the thread parameter. The rest of the elements are the remaining parameters to the action. All the parameters have the syntax (type ?name), where type is the type and ?name is the name. Note that all variable names in TDDL start with a "?".
- ":split-precond": These are requirements in the form of logical expressions that need to hold for the action to be used. Furthermore, the order of the preconditions need to be the same as the order they are specified in the parameter list, except for the agent variable which is left out. More importantly, TFPOP assigns variables in the same order as they are specified. Hence, it is possible to decrease the search time by having the most restricting variable first in the argument list and the split precondition. A final note regarding the split precondition is that a precondition cannot use any of the parameters that are after itself in the parameter list since they are not yet bound to a value.

```

76 (:operator :onlypart (deliver (uav ?uav) (thread ?thread) (crate ?c)
77                               (location ?loc))
78   (:split-precond
79     ?thread => (:true)
80     ?c => (and (not (= ?c null-crate)) (= (holding ?uav) ?c))
81     ?loc => (= (at ?uav) ?loc)
82   )
83   (:phase
84     (:duration 1 3 8)
85     (:effects
86       (:= (at ?c) ?loc)
87       (:= (holding ?uav) null-crate)
88     )
89   )
90   (:always-followed-by)
91   (:definitely-changes (at ?c) (holding ?uav))
92   :can-achieve-goal
93 )
94
95 (:operator (land (uav ?uav) (thread ?thread))
96   (:split-precond
97     ?thread => (= (is-flying ?uav) true)
98   )
99   (:phase
100     (:duration 2 5 7)
101     (:effects
102       (:= (is-flying ?uav) false)
103     )
104   )
105   (:always-followed-by)
106   (:definitely-changes (is-flying ?uav))
107   :can-achieve-goal
108 )

```

Figure 2.7: The people-in-distress domain: operators part 2.

- `:phase`: This what the that the action does. It consists of three different parts. The first part is how long time the action takes. This is written with the `:duration` keyword which requires three different durations: the minimum, the expected and the maximum duration for the action. The second part is the changes in relations and properties that the action make. These are specified with the `:effects` keyword and simply lists all new relations and properties. Note that this will overwrite the previous value of the relations and properties. The third part is changes in resources. However, as mentioned before, resources are not needed for this report and will therefore be left out.
- `:always-followed-by`: This is a flag for the action which specifies which actions that are allowed to follow the action. The actions are specified as a list of names.
- `:definitely-changes`: This is a list of fluents that the action promises to change.
- `:can-achieve-goal`: This is a boolean flag that, if it exists, states that this action can achieve a goal for at least one problem instance that uses the domain. Hence, TFPOP should check if the goal is fulfilled after this action is applied.

The final part of the domain model (figure 2.8) shows the sequence action. This part is not a required part of the domain. However, it can provide useful domain knowledge that TFPOP can used to decrease the search time. A sequence action includes the following parts:

- `Naming, only part flag and parameters`: This has the same syntax and semantics as for normal operators except that it starts with `:composite-sequence` instead of `:operator`.
- `:split-precond`: This has the same syntax and semantics as for normal operators.
- `:actions`: This specifies which primitive actions that is to be executed in the sequence action. It is written as a list of primitive actions with the variables that are going to be used as the arguments.
- `:stnu`: This is a list of time constraints that must hold in the plan. Each time constraint starts with the keyword `:requirement` (or `:contingent` but this is not required for the report and is therefore left out). Following that is the start node of the constraint. After


```

110 (:composite-sequence (deliver-and-return (uav ?uav) (thread ?thread)
111                                     (base ?base) (crate ?what)
112                                     (people ?to) (location ?loc))
113   (:split-precond
114     ?thread => (and (= (holding ?uav) null-crate)
115                  (= (is-flying ?uav) false))
116     ?base => (= (at ?uav) ?base)
117     ?what => (and (not (= ?what null-crate)) (= (at ?what) ?base))
118     ?to => (:true)
119     ?loc => (= (at ?to) ?loc)
120   )
121   (:actions
122     (pick-up ?uav ?thread ?base ?what)
123     (move ?uav ?thread ?loc)
124     (deliver ?uav ?thread ?what ?loc)
125     (move ?uav ?thread ?base)
126   )
127   (:stnu
128     (:requirement start#3 [2, 9] end#3)
129   )
130   (:always-followed-by)
131   (:definitely-changes (at ?what))
132 )
133 )

```

Figure 2.8: The people-in-distress domain: sequence operators.

that comes the time constraints expressed on the form " $[t_0, t_1]$ ". Finally, there is the end node of the constraint. In essence this means that there must pass between " $[t_0, t_1]$ " time units between the two nodes. Nodes are written as "start#x" for invocation node of action x in the list or "end#x" for effect node of action x in the list (the list is 0-indexed).

- ":always-followed-by", ":definitely-changes": These have the exact same syntax and meaning as for normal actions.


Problem specification

A problem instance specifies which objects exist, which relations and properties hold for these objects in the initial state, which agents exist and what the goal state is. Just like in the domain, the syntax for the problem instance will be explained using the people in distress example. The problem instance is shown in figure 2.9 and consists of the following:

- ":tfpop-problem": This part of the problem says that it is a problem instance for TFPOP named "distressed-people-at-mountain".
- ":domain": This specifies which domain that the problem instance exists in. In this example, it states that the domain is the "people-in-distress" domain.
- ":agents": This part specifies which agents exist in the problem instance. It is a list on the form (type (name thread*)) that says which type the agent has, what the name of the agent is and which threads it has.
- ":objects": This is a list of all the problem specific objects and their types.
- ":init": This part states the values are in the initial state for all the fluents in the domain. One way to do this is to manually assign values to all fluents, like in line 19 in figure 2.9. However, this can become quite a lot to write if there are a lot of objects. Instead, there is the possibility to use the "all" keyword which specifies that all instances of a fluent should have the specified value, like in line 16 in figure 2.9. Note that all fluents need to be specified in this part in order to give TFPOP full knowledge about the initial state.
- ":goal": This is the last part of the problem specification which states what should be true for the problem instance to be solved. The syntax for this is a list of predicates that follows the ":goal" keyword.

```
1 (:tfpop-problem distressed-people-at-mountain
2   (:domain people-in-distress)
3
4   (:agents
5     (uav (uav1 uav1-thread))
6     (uav (uav2 uav2-thread))
7   )
8   (:objects
9     (crate empty-crate supply-crate)
10    (base home-base)
11    (location isolated-mountain)
12    (people distressed-people)
13  )
14
15  (:init
16    (:all at home-base)
17    (= (at distressed-people) isolated-mountain)
18    (:all is-flying false)
19    (:all holding null-crate)
20    (:all is-empty true)
21    (= (is-empty supply-crate) false)
22  )
23
24  (:goal
25    (= (at supply-crate) isolated-mountain)
26    (= (at uav1) home-base)
27    (= (at uav2) home-base)
28  )
29 )
```

Figure 2.9: The distressed people at mountain problem instance.



3 Theory

The theory chapter begins with a section introducing notations and definitions that will be used through the report. Following that is a section covering composite actions in planning and logic. Thereafter, comes a section presenting different ways to measure understandability.

3.1 Definitions and notations

The first thing that needs to be established is what actions are in this report. Actions are divided into two categories: primitive actions and composite actions. Primitive actions are actions that are atomic in a sense that they do not depend on any other action. In a TDDL, the primitive actions are defined with the ":operator" keyword. Composite actions are all the actions that are dependent on at least one other action. The composite actions can be created by using different types of constructs called composite action components in this report. For example, there are control constructs, like while and if, and time constraints constructs.

In the report, a single set of notations will be used to increase clarity. This means that some formulas presented will be altered to fit the notation. The notation is presented in table 3.1.

3.2 Composite actions

The idea of composite actions is not new. One example is the logic programming language Golog from 1997 which makes use of complex actions and procedures that are similar to composite actions [15]. Furthermore, composite actions have been used in planning before. For example, Baier, Fritz and McIlraith defined a language based on Golog that can be compiled together with a domain [1]. The result is a domain and problem instance that includes all composite actions defined in their language. These two and more that relate to composite actions will be presented in the following sections.

Golog

Golog is a logic programming language based on situation calculus. Each program written in this language requires a set of axioms that model the domain that the program is to be

| Notation | Meaning |
|---------------|--|
| a | An action |
| η | The name of an action |
| ω | A composite action component |
| i and j | Integers |
| x and y | Variables |
| t | A variable for a time point |
| T | A type of variable |
| ϕ | A boolean formula |
| $[t]\phi$ | A boolean formula at a given time point |
| $[t_1, t_2]a$ | An action that takes place between t_1 and t_2 |

Table 3.1: Notations used in the report.

executed in [15]. The axioms state which primitive actions exist, what their preconditions and effects are, which fluents exist and the initial state. The program itself contains a set of named procedures followed by a start procedure. These procedures correspond to composite actions. Furthermore, the procedures consist of the following complex actions (which corresponds to composite action components) [15]:

- A test action which creates an action with only preconditions from a boolean formula ($\phi?$).
- A sequence of two actions ($[a_1; a_2]$).
- A nondeterministic choice of two action ($((a_1|a_2))$).
- A nondeterministic choice of argument for an action ($((\pi x)a(x)$ where π is an operand that selects the argument x nondeterministically and action a is an action which takes one argument).
- Iteration over an action zero or more times (a^*).
- Conditions (*if ϕ do a_1 else a_2 endIf*).
- While-loops (*while ϕ do a_1 endWhile*).
- Recursion through named procedures (*proc $\eta(x_1, x_2, \dots, x_i)a$ endProc*; where a can include call to η).

A drawback with the original Golog is that it does not support concurrent actions [2]. However, a derivation of Golog called ConGolog includes a complex action that does not specify which of two actions happen first [7]. It was later extended to include a complex action ($a_1||a_2$) that allows for two actions to be concurrent in a sense that they can be applied at the same time or one by one in any order in TConGolog [2].¹

Procedural domain control knowledge

Based on Golog, Baier, Fritz and McIlraith defined a language (which was never named but will be called BFM in this report) for expressing domain knowledge [1]. The domain knowledge in a BFM program can be compiled with a domain written in PDDL2.1 to a new domain and problem instance in PDDL2.1. This domain and problem instance contain the domain knowledge expressed in the BFM program and can be used for all the problem instances in the original domain.

¹The article presenting ConGolog was published in 2000 and the paper describing the TConGolog was published in 1999. However, the submission of the article for ConGolog was available in 1999. Hence, TConGolog was developed later even though it was published earlier.

The syntax of BFM is defined in terms of programs instead of actions [1]. All the components (corresponding to composite action components) that a BFM program can consist of [1] is presented in the following list:

- A empty action (*nil*).
- A nondeterministic choice of action (*any*).
- A test action, working in the same way as in Golog ($\phi?$).
- A sequence of two actions ($a_1; a_2$).
- Conditions (*if ϕ then a_1 else a_2*).
- While-loop (*while ϕ do a*).
- Iteration over an action zero or more times (a^*).
- A nondeterministic choice of two actions ($a_1|a_2$).
- A nondeterministic choice of an argument with a specified type for an action ($\pi(x - T)a(x)$ where π is an operand that selects x nondeterministically from all variables with the type T).

Temporal composite actions

Temporal Action Logic (henceforth, called TAL) has been extended with composite actions by Doherty, Kvarnström and Szalas [8]. The extension includes a way of expressing composite actions with constraints. This is done through defining a composite action as the following (note that this is an adapted version to describe the concept of the composite action, not the formal definition):

$$[t_1, t_2]\eta(y_1, y_2, \dots, y_i) \rightsquigarrow \text{with } x_1, x_2, \dots, x_j \text{ do } \omega \text{ where } \phi$$

The formula means that ω is executed sometime between the time points t_1 and t_2 . Furthermore, the variables used in ω are the parameters y_1, y_2, \dots, y_i and the newly introduced x_1, x_2, \dots, x_j where x_1, x_2, \dots, x_j can be chosen freely from all variables as long as ϕ is true with the chosen variables. Moreover, ω is defined as one of the following (corresponding to composite action components) [8]:

- Sequence of actions ($(a_1; a_2)$).
- Concurrent actions ($(a_1||a_2)$).
- Conditions (*if $[t]\phi$ then a_1 else a_2*).
- While-loops (*while $[t]\phi$ do a*).
- Concurrent loops over variables (*foreach x_1, x_2, \dots, x_i where $[t]\phi$ do conc a* , which means to do a for each vector of variables x_1, x_2, \dots, x_i under which $[t]\phi$ is true).
- Recursion trough named composite actions (see the formula above named η).
- Nondeterministically choose variables ($[t_1, t_2]\text{with } x_1, x_2, \dots, x_i \text{ do } \omega \text{ where } \phi$).
- Temporal constraints (see formula above).

Hierarchical task network

Hierarchical task network (henceforth, called HTN) is, as mentioned in the background chapter, a type of planning. Most important for this report is that HTN uses something called high-level actions (henceforth, called HLA) which is a sequence of primitive actions and HLA [17]. This means that they are essentially using composite actions but only what have been called sequence of actions in the previous sections ².

²HTN also includes the possibility of recursion since the HLAs are named. Unfortunately, this was missed and the mistake was only noticed at the end of the project. Therefore, the recursion is not mentioned before the discussion in this report.

3.3 Understandability measurement

There is no universally accepted definition of what is included in understandability [3]. Therefore, the first section covers a definition of understandability that will be used in this report. Following that is a section covering metrics for measuring understandability. The final section covers measurements that can be used to measure understandability in experiments.

Defining understandability

The definition used in this report is an adapted version of the definition that Bansiya and Davis use in their quality model for object-oriented design [3]. The adaptation is a simple change of words from design properties to domain model, resulting in the following definition:

Definition 1. *Understandability are the properties of the domain model that makes it easy to learn and understand.*

Software metrics

There are a lot of metrics that aim to measure the understandability of software. Hence, it is not possible to present all of them here. Instead, a selected few are presented in the following sections.

Quality Model for Object-Oriented Design

Quality Model for Object-Oriented Design (called QMOOD) is a hierarchical model for measuring the software quality of an object-oriented design [3]. The model breaks up software quality into multiple high-level attributes. Each of the high-level attributes is in turn mapped to a set of object-oriented design properties. These design properties has a metric that can be used to measure them. Weighting the metrics together gives a measurement for each of the high-level attributes [3].

The interesting part for this report is that one high-level quality is understandability. This quality is measured in the model using the following metrics and weights [3]:

- Abstraction (*weight*= -0.33): The average number of classes a class inherits information from.
- Encapsulation (*weight*= 0.33): The average percentage of all the attributes in a class that are private or protected.
- Coupling (*weight*= -0.33): The average count of all classes that are related through attributes and parameters to a class.
- Cohesion (*weight*= 0.33): The average of the following calculation for all the classes in the design:

$$\frac{\sum_{m \in M} p_m}{P * n}$$

Where p_m is the number of distinct parameter types in method m and M is the set of all the methods. Furthermore, P is the number of distinct parameters for all the methods in the class and n is the number of methods in the class.

- Polymorphism (*weight*= -0.33): Measured as the average count of polymorphic methods.
- Complexity (*weight*= -0.33): Measured as the average of the count of all the methods in a class.
- Design Size (*weight*= -0.33): Measured as the number of classes.

Code and data spatial complexity

The code and data spatial metrics are based on the limits of the working memory [6]. The main idea behind the metrics is that it is harder to understand the software if it puts a heavier load on the working memory. Hence, the longer it is between a definition and the use of a module (this could for example be a function, method or a sub-program) and the longer it is between the uses of a data member the harder it becomes to understand [6]. Based on this, two metrics are proposed (the formula for both can be found in appendix A.1).

Firstly, the code-spatial complexity metric that aims to measure the effort it takes to understand how the different modules in the software are connected [6]. The metric is calculated according to the following:

1. Calculate the average distance from module definition to the module use for all the modules in the software.
2. Calculate the average of all the average distances for all modules.

Secondly, the data-spatial complexity metric that aims to measure the effort it takes to understand what the value of a data member is. In this metric, it is more interesting to look for the previous value than the definition since the value of the data member can change [6]. Moreover, Chhabra, Aggarwal and Singh found that local data members within a module do not have an impact worth mentioning and were therefore excluded [6]. As a result, the metric is calculated as follows:

1. Calculate the average distance to the use of a global variable from where it was last used for all global variables.
2. Calculate the average of all the average distances for all the global variables.

Improved cognitive information complexity

The improved cognitive information complexity measure is a metric of software understandability with its root in informatics and cognitive science [13]. It is based on the that software can be treated as information. Therefore, difficulty in understanding software is equal to difficulty in understanding information. Based on this, the metric is calculated with the following steps [13] (the exact calculation can be found in appendix A.2):

1. Calculate the information content based on the operators and identifiers for each code line.
2. Calculate the effect each code line has on the rest of the software using its line number and the total number of lines of code.
3. Sum the impact for all the lines of code.
4. Calculate the sum of the cognitive weight of all the control structures as presented by Wang and Shao [20].
5. Multiply the summed cognitive weight of the control structures with the summed impact.

Experiments

Measuring understandability by performing experiments with users or experts is an alternative that does not require a validated metric. The following sections cover different types of measurements that have been used to measure understandability in experiments. Furthermore, there is a section about measuring something through indirect questions.

User estimation

One way to measure the understandability of a piece of software is to allow the user to estimate the understandability of the software. This has been done by Harrison, Counsell and Nithi to assess what effect inheritance in object-oriented system has on understandability [10] and by Genero, Poels and Piattini to validate metrics for understandability of entity relationship diagram [9].

Both of the above mentioned studies followed similar experimental design. The participants were presented with some questions aimed at making the participants understand the code respectively the entity relationship diagram [9, 10]. Thereafter, the users were asked to estimate how understandable the code respectively the entity relationship diagram using a labeled scale from one to five. The answer for this is the measurement for understandability.

Time and correct answers

Genero, Poels and Piattini measured the time it took to answer the questions about the entity relationship diagram that was presented in the experiment mentioned previous section [9]. This was done in addition to having the participants estimate the understandability of the diagrams. The time was measured by prompting the participants to write down the time before they started to answer the questions aimed at making them understand the diagrams and directly after having answered them.

In addition to measuring the time, Genero, Poels and Piattini used the answers to the questions to create two more measurements [9]: The number of correct answers; and the number of correct answers divided by the time the participants took to answer the questions. These two values were used for the validation of their model, in addition to using the users estimated understandability.

Expert ranking

A possible way to measure understandability is to let experts order a set of items (in this report a set of domains). It is then possible to read if an item is rated as having a higher understandability than another. This is the method that Bansiya and Davis used to validate their metric for object-oriented design [3]. In their experiment, they had experts rank 14 designs and used their metric to rank the same designs. The two rankings were then statistically tested for any correlations.

Indirect questions

The final method that can be used to measure understandability that will be covered is by measuring it through indirect questions. It is similar to user estimation in most ways. However, the difference lies in using questions based on the different aspects of understandability to measure understandability. This is something that the widely used System Usability Scale uses [5]. Moreover, Boehm, Brown and Lipow identified five aspects of software that affect understandability [4]. These are presented in table 3.2.

| Aspect | description |
|----------------------|--|
| Conciseness | The software is concise if it does not contain excessive information. |
| Consistency | The software is internally consistent if it has uniform notation and terminology and external consistent if the notation and terminology conform with external requirements. |
| Legibility | The software is legible if its function can be understood from reading it. |
| Self-descriptiveness | The software is self-descriptive if it contains enough information to understand its meta-information (for example its objectives and assumptions). |
| Structuredness | The software is structured if it is organized according to a pattern. |

Table 3.2: Software understandability aspects according to Boehm, Brown and Lipow [4].



4 Method

The method chapter is divided into five parts. This is because the first and the third research questions each has a sub-question that needs to be answered before the research question can be answered. Hence, the first section describes how it will be decided which composite action components that are going to be implemented in TFPOP. The section after that covers how the implementation will be done. Following that is the method for how the questions regarding search time will be answered. Thereafter, is a section describing how a measurement method for understandability will be selected. The final section covers the test whether composite actions can be used to increase the understandability of a domain.

4.1 Composite action component ranking

The question regarding which composite action components to implement must be answered before the implementation of composite actions can be done. This question will be answered by categorizing the different composite action components in publications using composite actions. These categories will then be ranked according to the number of publications that include a component from that category. The resulting ranked list will be used as a foundation for which composite action components that should be implemented. In addition, the composite action components in the current implementation of sequence action is prioritized so that the current functionality will be present in the implementation. Finally, the number of composite action components that are implemented is restricted by the time frame of the report.

4.2 Implementation

In the implementation, all the selected composite action components will be implemented. Moreover, the implementation will aim to be as easy as possible to extend with other composite action components. This is because not all the composite action components will be implemented. Therefore, it is valuable to make it as easy as possible to extend the implementation with more composite action components.

4.3 Search time

There are two sub-questions regarding search time. As a result, two experiments will be performed. These follow the guidelines presented by Kitchenham et al. [11] with some modifications. The modifications are justified with that there are no human participants in these experiments and therefore there is no need to handle any effects that might affect the participants.

Compared to no guidance

The search time has to be measured in order to answer the question if composite actions can be used to decrease search time. This will be done by measuring the number of milliseconds of CPU usage that TFPOP uses to find an optimal solution.

There is one problem with comparing search guidance with composite action to no search guidance at all. The problem is that the depth first search that TFPOP usually does can in theory take infinite time to conduct. To prevent this, minimal search guidance can be added by restricting the depth of the search tree. This does not affect the order in which TFPOP selects the actions that is within the allowed depth. Therefore, the search time when restricting the depth is less or equal to the search time when doing an unrestricted search. Hence, proving that using search guidance with composite actions is faster than search guidance that restricts the depth also proves that it is faster than using no search guidance at all.

Gathering the data will be done by having TFPOP find an optimal solution to a problem instance (see appendix B figure B.5) using two different domains. One domain that contains a composite action (see appendix B figure B.2) and one domain that does not contain any extra domain knowledge (see appendix B figure B.1). Both domains model the Blocks World with one primitive action that moves a block from one position (on a block or on the floor) to another. To enforce that TFPOP uses the composite action, the primitive action has the onlypart flag in the domain that has a composite action. In addition, the depth for the search in both cases have been restricted to four (the optimal solution uses three actions).

Data points will be gathered for each domain twenty times in row. This is to generate a sample for the search time for TFPOP to find an optimal solution to the problem instance for both domains on the testing computer (MacBook Air running OS X El Capitan, version 10.11.4, with 1.3 GHz dual core and 4GB RAM). Before any data was gathered for a domain TFPOP will have to solve the same problem instance in that domain twenty times. This will be done to minimize the effect of that JVM optimization during run-time has on the results.

The analysis of the data will be done with a one-tailed t-test using a between group design. Levene's test will be used to test for equal variance and Cohen's d will be used to measure the effect. The hypotheses for the test are the following:

- H_a : The search time for finding an optimal solution is lower when TFPOP uses the domain that has a composite action.
- H_0 : The search time for finding an optimal solution is equal or greater for TFPOP when it uses the domain that has a composite action.

A successful rejection of H_0 means that using a composite action is significantly (in the statistical meaning) faster for this problem instance and thereby allowing the conclusion that composite actions can be used to decrease search time.

Compared to equivalent guidance

Finding an indication of whether search guidance with composite actions can be as good as the equivalent search guidance without composite actions will be done using the Blocks World domain. There are two domain specifications that will be used in this experiment. Firstly, a domain that uses composite actions which is the same as the one in the previous

section (see appendix B figure B.2). Secondly, a domain that has extra domain knowledge that is equivalent to that of the composite actions used in the first domain (see appendix B figure B.3 and B.4). The equivalence means that extra primitive actions that force TFPOP to go through the same steps as in the composite actions (check conditions in if and while and applying primitive actions) are included. In essence, this means that multiple primitive actions are added to the domain along with fluents and constants that ensure that the extra actions can only be used in the same order as the composite actions. For this experiment, it requires that the domain has multiple extra primitive actions and two extra fluents to keep track of the state of what should be done next and which values to use for the parameters to the primitive actions. Furthermore, the primitive action will be marked as `onlypart` in both cases to ensure that the extra domain knowledge is used by TFPOP.

This experiment will be using 100 problem instances containing between 2 and 101 blocks (each problem instance has a unique number of blocks). The goal of all the problem instances is to put all the blocks on the floor and the initial state is partly randomized. The non randomized part is that the initial state will always have the first block placed on the floor and the second block placed on top of the first to ensure that at least one action is needed. This will be done since TFPOP does not check if the initial state fulfills the goal. However, the rest of the blocks will be placed, randomly, on the floor or on any of the previous placed blocks that still are clear. Moreover, there will be two versions of each problem instance with only one difference. The difference is that the problem instance for the domain that does not use composite actions has to initialize its extra fluents. An example of the problems can be found in appendix B figure B.6 for guidance with composite actions and figure B.7 for equivalent guidance without composite actions.

The data gathered in the experiment is the search time measured in milliseconds of CPU usage. Moreover, the recorded data is an average of three consecutive executions for each problem instance and domain. This will be done to reduce the effect that fluctuations in search time for a problem instance have on the results. Otherwise, the gathering of data will be done in the exact same way as in the experiment in the previous section. This includes that each problem instance has to be solved twenty times for a domain before the data points are gathered for that problem instance and domain. Hence, each problem instance will be solved 23 times where the three last will be used to measure the average search time for a problem instance.

The analysis of the results will be an observation of the gathered data points. No further analysis will be performed since the result is only used as an indication.

4.4 Understandability measurement

A fundamental requirement for being able to evaluate the changes in understandability is to measure it. There are two major ways of doing this. The first is to calculate the understandability from code and documentation according to some metric and the second is to conduct an experiment. This part of the report aims to find out how the measurement of understandability is going to be performed in the understandability experiment.

The method for finding measurements is an informal literature study of existing measurements, including but not restricted to metrics and experiments. The requirements for a viable measurement are the following:

- It is possible to use the measurement on a domain file written in either PDDL or TDDL. This include measurements that are directly applicable and measurements that can be adapted without creating validity problems.
- The measurement should be established. This is based on the number of citations the articles have since this is an indication of how many have used the measurement. The minimum limit of required citations was decided to be 100.

- The measurement needs to be usable with the resources that are available. This includes time limits and other resource limits.

The search for references will be done through three methods. The first is a simple keyword search on *www.scholar.google.com*. This will be done by searching on combinations of keywords from the categories presented in table 4.1. The second is to look at the references in the found articles. These two methods will be used concurrently with the second providing keywords to the first and the first providing articles to the second. The third method is to dig a bit deeper in a measurement method that is looking like it can meet the requirements. This includes looking for the same measurement method when it is not used for measuring understandability. The whole procedure is described in figure 4.1.

| Category | Keyword |
|--------------------|----------------------------------|
| What to measure | Understandability |
| | Maintainability |
| | Readability |
| | Cognitive information complexity |
| Measurement target | Software |
| | Program |
| | Code |
| Measurement type | Metric |
| | Measurement |

Table 4.1: All the search keywords sorted after categories.

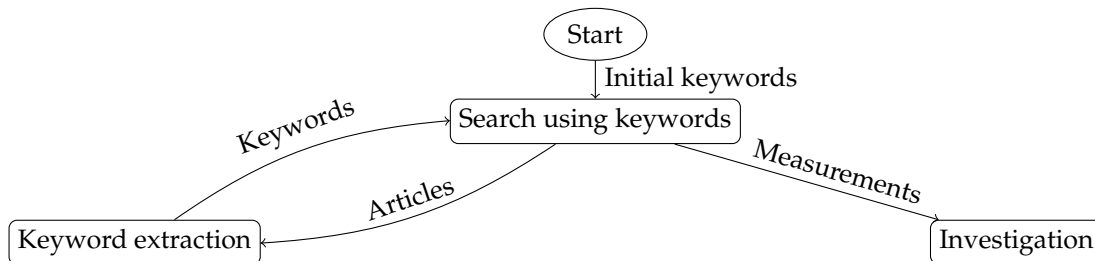


Figure 4.1: The method for searching for understandability measurements.

The choice of which measurement(s) to use in the understandability experiment is based on the requirements presented above. In case that multiple measurements meet the requirement, a selection based on the popularity of the measurements within the field of planning and their credibility for this study will be used.

4.5 Understandability experiment

Based on the results for understandability measurements, a user experiment will be conducted to evaluate if understandability can increase by using composite actions in the domain model. The experiment has a within-subjects design (compared data points are measurements from the same subject under different conditions) and is based on the guidelines for empirical research by Kitchenham et al. [11].

The following sections cover the method for the experiment. First is a section that presents the questionnaire that will be used to gather data. Next is a section describing the experimental unit and the measurements in the experiment. After that is a section covering a pre-experiment for estimating the needed sample size and to test the questionnaire. Following

that, is a description of how the experiment is set up and will be conducted. Thereafter, is a section about how the data will be analyzed. Finally, there is a section covering how bias and other unwanted factors that can affect the result are handled.

Questionnaire design

For measuring the understandability there needs to be a medium through which it is measured. The medium in this experiment is a questionnaire just like in Harrison, Counsell and Nithi's experiment [10] and Genero, Poels and Piattini's experiment [9] regarding understandability in object-oriented system respectively entity-relationship diagrams. The design of the questionnaire in this study is based on their experiment and on the recommendations presented by Krosnick and Presser [12] in *Questions and Questionnaire Design*.

Two decisions that affected the design for the questionnaire were taken. The first is to use PDDL instead of TDDL for the domains (with similar syntax for composite actions). This is because there is only a handful of persons that know TDDL at the moment and all of them are involved in this report in some manner. Moreover, both of them are similar so that the results on PDDL should be applicable on TDDL. The second is to write the questionnaire in Swedish. This was decided to minimize the effect of any language barrier both when formulating and interpreting the questionnaire.

The design of the questionnaire is described in the two following sections. The first covers the general layout of the questionnaire. That includes the order of questions, information given to the participant, et cetera. The second part covers the formulation of questions that aim to measure understandability.

Questionnaire layout

The general layout of the questionnaire is based on the questionnaires used by Genero, Poels and Piattini [9] and Harrison, Counsell and Nithi [10]. Both of these questionnaires presented the participants with an item (entity relationship diagram respectively c++ code) followed by some questions that aim to make the participants understand the item. After that, the participants were presented with a question regarding the understandability of the item. This layout is adapted to work with the within-subject design of this experiment by presenting two items to the participants. Moreover, neither of the experiments presented anything unknown to the participants [9, 10]. Therefore, the layout was modified to include an introduction to composite actions. This resulted in the following layout for the questionnaire:

1. An introduction to how the experiment is conducted.
2. An introduction to composite actions in PDDL.
3. The first domain file with questions:
 - a) Instructions for what the participant is supposed to do.
 - b) A question to fill in the current time.
 - c) The PDDL file.
 - d) Questions aimed to make sure that the participant has understood the PDDL file.
 - e) A question to fill in the current time.
 - f) Questions aimed to measure the understandability.
4. The second domain file with questions:
 - a) Instructions for what the participant is supposed to do.
 - b) A question to fill in the current time.
 - c) The PDDL file.
 - d) Questions aimed to make sure that the participant has understood the PDDL file.
 - e) A question to fill in the current time.
 - f) Questions aimed to measure the understandability.

Formulation of understandability questions

The formulation of the questions regarding understandability were based on the recommendations presented by Krosnick and Presser [12]. These recommendations include that the questions should not be negations and they should not be leading. Furthermore, all data points on an answer scale should be labeled and have equal distance between them. In addition, an answering scale of length five has both high validity and reliability [12]. Finally, using questions that is not agree/disagree questions increase the reliability and validity [12]. Based on this, a fully labeled answering scales of length five are used for all the questions. Furthermore, non of the questions are agree/disagree questions. Moreover, each question is used in two versions: one that is formed positively (for example "how easy[...]") with the negative end of the scale as the first option; and one that is formed negatively (for example "how misleading[...]") with the positive end of the scale as the first option.

For measuring the understandability both user estimation and indirect questions are used. The user estimation consists of two questions (one positive and one negative) regarding the participants' estimation of the understandability. Furthermore, all but one of the aspects of understandability according to Boehm, Brown and Lipow was used as a basis for the indirect questions [4]. The aspect that is not used is conciseness since the whole idea with composite actions is to include more information in the domain. Hence, conciseness could easily be misleading. Nevertheless, the four remaining aspects were used in one positive and one negative question each. However, it is not validated that these questions measure understandability. Therefore, the result from the indirect questions will only be used as an indication. Finally, the order of the questions were shuffled so that no positive and negative question regarding the same thing was adjacent to each other.

Experimental unit and measurements

The experimental unit for this experiment is the individual. Moreover, for each of the experimental unit two measurements will be calculated. To compute the measurements, all the answers will be given a number from 1 to 5 where 1 corresponds to the most negative response and 5 to the most positive response. Using these numbering the following measurements will be calculated:

- *Estimated understandability*: The participants estimated understandability of a PDDL file. This is calculated as the average of question 1 and 6. Hence, this value can be between 1 and 5.
- *Measured understandability*: This variable measures the understandability of the PDDL file through the indirect questions. The calculation of this is done by taking the average of questions 2-5 and 7-10. Hence, the measurement has a value between 1 and 5. Note that this is the non validated indirect questions and therefore this measure will only be used as an indication.

Pre-experiment

A pre-experiment will be conducted with two purposes. The first is to give an estimation of the needed sample size for significant result in the experiment as recommended in the guidelines by Kitchenham et al. [11]. This will be done by calculating the needed sample size for a paired t-test where the mean and standard deviation are from the gathered data. Estimating the sample size in this way is not an exact measurement but it gives an indication of the needed sample size. The second is to evaluate the questionnaire to make sure that the questions are well formulated as recommended by Krosnick and Presser [12].

The pre-experiment will be conducted by giving the participants a short introduction to PDDL (see appendix D). This is done since the participants will be students at Linköping University who took a course in artificial intelligence half a year ago. The reason to give them

an introduction to PDDL is to allow them to refresh their knowledge about PDDL. Thereafter, they will be given the questionnaire and asked to fill it out. The questionnaire will have the domain with composite actions first for every second participant, starting with the first, and the domain with the composite actions last for the other participants.

Questionnaire updates

The pre-experiment showed that some questions aimed at making the participants understand the domains were hard to understand. As a result, these questions were updated. Moreover, measuring the time was also removed from the questionnaire.

The experiment

The participants in the experiment will be students at Linköping University that are currently taking a course in automated planning and freely agree to participate in the study. Therefore, the population for the experiment is students at Linköping University with at least basic knowledge about planning.

As in the pre-experiment, the different treatments are whether the domain that has composite actions is the first or the second domain in the questionnaire. The treatment that the participants get only depends on the order in which they participated.

The conduction of the experiment will be done by supplying the participants with a questionnaire and ask them to fill it out. Unfortunately, there is a risk that the sample size will be smaller than what the pre-experiment indicates as needed for a significant result. If that is the case, then the questionnaire will be handed out to the students during lab sessions and lectures in the course in automated planning. They will be asked to fill it out when they can but to do it without taking any breaks. The reason to not take any breaks is to minimize the effect of unknown variables. For example, using different internal scales when answering questions for the first and the second domain.

Data Evaluation

The results from the questionnaires will be evaluated by testing for a significant difference in the measurements between the two items. Testing for a significant result will be done with a one-tailed Wilcoxon signed-rank test. Furthermore, the effect will be measured using Pearson's r . The hypotheses for this experiment are the following:

- $H_{(a,1)}$: The domain file that has composite actions will have a higher understandability than the equivalent domain without composite actions, using *estimated understandability* as a measurement for understandability.
- $H_{(0,1)}$: The domain file that has composite actions has the same or lower understandability as the equivalent domain without composite actions, using *estimated understandability* as a measurement for understandability.
- $H_{(a,2)}$: The domain file that has composite actions will have a higher understandability than the equivalent domain without composite actions, using *measured understandability* as a measurement for understandability.
- $H_{(0,2)}$: The domain file that has composite actions has the same or lower understandability as the equivalent domain without composite actions, using *measured understandability* as a measurement for understandability.

A successful rejection of $H_{(0,1)}$ means that the domain file with composite actions has a significantly higher understandability under the assumption that *estimated understandability* measures the understandability of a domain file. In the same manner, a successful rejection of $H_{(0,2)}$ means that the domain file with composite actions has a significantly higher understandability given that *measured understandability* measures understandability.

Bias and unwanted effects

All experiments need to handle any potential bias and other unwanted effects that can affect the result of the experiment. Following the guidelines presented by Kitchenham et al. [11], the identified factors will be presented in the following paragraphs along with the steps taken to mitigate them. The different effects are not ordered by the magnitude they can have on the results.

The first effect that will be presented is the primacy and recency effect of a scale. These effects state that the participants is more likely to choose an answer that is at the beginning respectively at the end of the scale. The most common of these two in a questionnaire where the answer alternatives are presented visually is the primacy effect [12]. In the questionnaire, the answer alternatives of a positive and negative question were ordered in opposite ways. As a result, the primacy and the recency effects should both affect the result in a positive and negative way and therefore minimize the magnitude of the effect.

One more effect related to the questions is the response bias towards agreement. That means that the participants tend to want to agree with a statement more often than they want to disagree with it [12]. The item reversal method that is used in System Usability Scale [5] was selected to mitigate the effect of the response bias. Item reversal means that there are multiple questions that measures the same thing but some of them are reversed in a sense that agreeing with them give the lowest score instead of the highest.

A long questionnaire is a problem since the participants get more and more exhausted toward the end of the questionnaire [12]. Both this problem and the learning effect will be mitigated in the same way. The learning effect is that the participants learn how a task in an experiment is done and will therefore perform better on similar tasks in the experiment. Mitigation of these two effects is done by letting half of the participants answer a questionnaire in which the first domain is the one with composite actions and the other half answer a questionnaire where this is the second domain. The ordering of the two domains is the only difference between the two questionnaires.

In the analysis there is a bias to find significant result where there is non if the analyzer has some investment in the experiment [11]. The one who does the analysis of the data in this experiment is the author who has an obvious investment in the experiment. To prevent this from affecting the analysis, the methods that are going to be used for analysis were selected before the experiment was conducted.



5 Results

5.1 Composite action component ranking

The ranking of composite action components based on previous publications are presented in table 5.1. The ranking is based on the categorization of composite action components presented in tables 5.2, 5.3, 5.4 and 5.5.

All the composite action components that have a ranking of 3 or higher were selected to be implemented. Furthermore, $[t_0, t_1]$ was selected as well since it is already present in the implementation of sequence action. No more composite action components were selected due to the time constraints of the report.

| Rank | Composite action component |
|------|---|
| 4 | $(a_1; a_2; \dots; a_i)$ |
| 3 | <i>with</i> x_1, x_2, \dots, x_i <i>where</i> ϕ <i>if</i> ϕ <i>then</i> a_1 <i>else</i> a_2 <i>while</i> ϕ <i>do</i> a $\phi?$ $a_i \in \{a \dots\}$ |
| 2 | <i>recursion</i> $(a_1 a_2 \dots a_i)$ a^* |
| 1 | <i>nil</i> <i>foreach</i> x_1, x_2, \dots, x_i <i>where</i> ϕ <i>do conc</i> a $[t_0, t_1]$ |

Table 5.1: The ranking of the composite action components where the rank corresponds to the number of publications using them.

| Composite action component (notation) | Operator(s) |
|---|--|
| Test action ($\phi?$) | $\phi?$ |
| Sequence ($((a_1; a_2; \dots; a_3))$) | $(a_1; a_2)$ |
| Nondeterministic choice from set of action ($a_i \in \{a\dots\}$) | $(a_1 a_2)$ |
| Nondeterministic variable choice (<i>with</i> x_1, x_2, \dots, x_i <i>where</i> ϕ) | $\pi(x - T)a(x)$ |
| Repeat action (a^*) | a^* |
| Condition (<i>if</i> ϕ <i>then</i> a_1 <i>else</i> a_2) | <i>if</i> a_0 <i>do</i> a_1 <i>else</i> a_2 <i>endIf</i> |
| While-loop (<i>while</i> ϕ <i>do</i> a) | <i>while</i> a_0 <i>do</i> a_1 <i>endWhile</i> |
| Recursion (<i>recursion</i>) | Recursion |
| Concurrent actions ($((a_1 a_2 \dots a_i))$) | $(a_1 a_2)$ |

Table 5.2: The categorization of complex actions in Golog.

| Composite action component (notation) | Operator(s) |
|---|--|
| Empty action (<i>nil</i>) | <i>nil</i> |
| Nondeterministic choice from set of action ($a_i \in \{a\dots\}$) | $(a_1 a_2)$ & <i>any</i> |
| Test action ($\phi?$) | $\phi?$ |
| Sequence ($((a_1; a_2; \dots; a_3))$) | $(a_1 a_2)$ |
| Conditions (<i>if</i> ϕ <i>then</i> a_1 <i>else</i> a_2) | <i>(if</i> ϕ <i>then</i> a_1 <i>else</i> $a_2)$ |
| While-loop (<i>while</i> ϕ <i>do</i> a) | <i>while</i> ϕ <i>do</i> a |
| Repeat action (a^*) | a^* |
| Nondeterministic variable choice (<i>with</i> x_1, x_2, \dots, x_i <i>where</i> ϕ) | $\pi(x - T)a(x)$ |

Table 5.3: The categorization of components in BFM.

| Composite action component (notation) | Operator(s) |
|--|--|
| Sequence ($((a_1; a_2; \dots; a_i))$) | $(a_1; a_2)$ |
| Concurrent actions ($((a_1 a_2 \dots a_i))$) | $(a_1 a_2)$ |
| Condition (<i>if</i> ϕ <i>then</i> a_1 <i>else</i> a_2) | <i>if</i> $[t]\phi$ <i>then</i> a_1 <i>else</i> a_2 |
| While-loop (<i>while</i> ϕ <i>do</i> a) | <i>while</i> $[t]\phi$ <i>do</i> a |
| Foreach concurrent (<i>foreach</i> x_1, x_2, \dots, x_i <i>where</i> ϕ <i>do</i> <i>conc</i> a) | <i>foreach</i> x_1, x_2, \dots, x_i <i>where</i> $[t]\phi$ <i>do</i> <i>conc</i> a |
| Recursion (<i>recursion</i>) | Recursion |
| Nondeterministic variable choice (<i>with</i> x_1, x_2, \dots, x_i <i>where</i> ϕ) | <i>with</i> x_1, x_2, \dots, x_i <i>do</i> ω <i>where</i> ϕ |
| Time constraints ($[t_0, t_1]$) | Temporal constraints |

Table 5.4: The categorization of components in the extension of TAL with composite actions.

5.2 Implementation

Composite actions was implemented similar to the already existing sequence action. That means that it essentially is an action with a state that keeps track of the current position in the composite action. The difference between the two types of non-primitive actions are that composite actions consist of a list of composite action statements (the building blocks of a

| Composite action component (notation) | Operator(s) |
|---|-------------|
| Sequence ($((a_1; a_2; \dots; a_i))$) | Tasks |

Table 5.5: The categorization of components in HTN.

composite action, which are covered later in the chapter) in addition to primitive actions. Moreover, the state corresponding to a composite action is different so that it can handle the new structure.

In addition to implementing composite actions in TFPOP, TDDL has been extended to support composite actions. The syntax for them was based on the syntax of the sequence action and is presented in figure 5.1. In fact, the only thing that is different at this level is that the ":actions" property is exchanged for the ":body" property. The new property consists of a list of composite action statements.

```

1 (:composite-operator (name parameter-list)
2   (:split-precond)
3   (:body non-empty-composite-action-statement-list)
4   <(:stnu)>
5   <(:always-followed-by)>
6   <(:definitely-changes)>
7 )

```

Figure 5.1: The syntax for composite actions in TDDL where everything within "<>" is optional.

Introducing the composite actions also introduced more kinds of branch points that TFPOP needs to handle. These branch points have been categorized into two categories which will be presented in the following section. The section after that will cover the new state that keeps track of the evaluation of a composite action. Thereafter, is a section covering how the different composite action components are implemented in the composite action statements. Finally, there is a section describing how TFPOP conducts a search with the composite actions.

Composite action branch points

As mentioned in the previous section there are two different types of branch points that are present in composite actions. The first type of branch points are those that correspond to a primitive action and therefore adds nodes to the plan. The second type are branch points that do not correspond to a primitive action and therefore may not add any nodes to the existing plan. However, they create causal links and variable bindings that need to be attached to a node. Moreover, both types can get causal links and variable bindings from previous branch points.

The first category of branch points is the same as the sequence branch point. In essence, it finds a set of causal links that ensure that, given a set of causal links and variable bindings from the previous branch point, all preconditions of the primitive action hold. The resulting set of causal links is a union of the two sets of causal links. These causal links are thereafter, attached to an invocation node for the primitive action. Finally, the invocation node and a corresponding effect node are added to the plan. The steps involved in this are described in figure 5.2.

The second category differs in that it only adds nodes to the plan if there are no statements within its scope that is to be evaluated. As a result, a branch point from this category finds a set of causal links and possibly variable bindings that ensure that the conditions for the corresponding composite action component hold. The resulting causal links and variable bindings are thereafter, passed to the next branch point along with the input from the previous branch point if there is a statement to evaluate within its scope. Otherwise, a null action is created and its corresponding nodes are added to the plan. The null action has no preconditions or effects and its duration is zero. Finally, the causal links are connected to the invocation node that corresponds to the null action. The working of the branch points in this category is shown in figure 5.3.

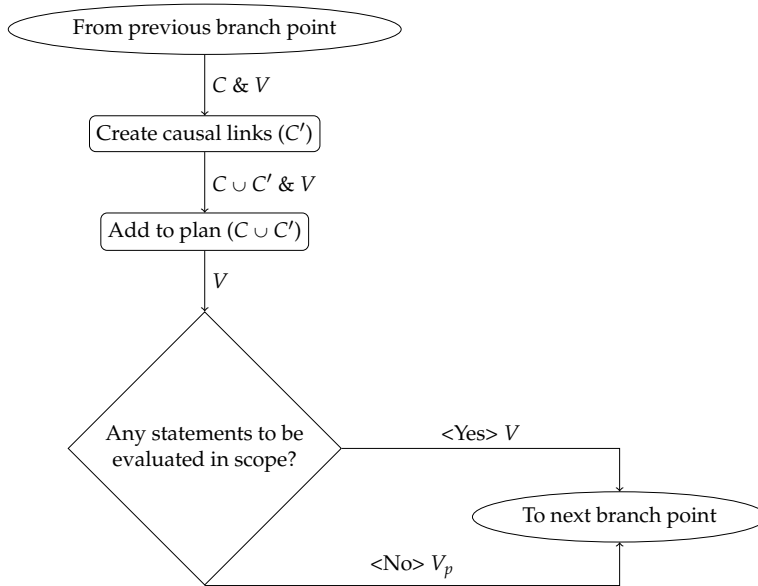


Figure 5.2: An overview of a composite action branch point that corresponds to a primitive action. C and C' are sets of causal links, V is a set of variable bindings and V_p is the variable bindings from the parents scope. Furthermore, " $\langle \rangle$ " indicates the decision at a decision node.

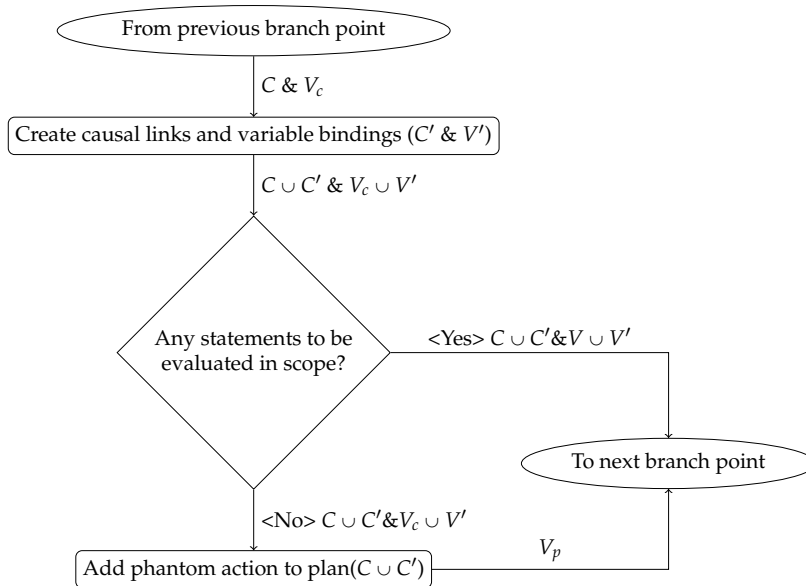


Figure 5.3: An overview of a composite action branch point that does not add any nodes to the plan. C and C' are sets of causal links, V and V' are sets of variable bindings and V_p is the variable bindings from the parents scope. Furthermore, " $\langle \rangle$ " indicates the decision at a decision node.

One more thing that should be mentioned about passing causal links and variable bindings is the scope. Each branch point has a scope of composite action statements to which corresponding branch points it can pass causal links and variable bindings to. In essence, this scope includes all composite action statements defined within it. Hence, only the variable bindings it got from its parent can be passed to branch points that correspond to composite action statements outside its scope. For example, imagine that a composite action consists of two composite action statements, x and y . x is the first one and therefore gets the causal

links and variable bindings from the precondition and parameter list of the composite action. Furthermore, x adds some more causal link with a branch point of the second category. However, the next branch point in the search is the first branch point in y which is outside the scope of x . Therefore, the branch point in x will create a null action to which the causal links are added. The corresponding nodes will then be added to the plan. Hence, no causal links are passed to as input to y .

Composite action state

TFPOP uses a state, called composite action state, to keep track of the evaluation of a composite action. However, this state is more complex than that of the sequence actions. This is because composite actions have more functionality.

To meet the requirements, the composite action states contain a Composite Action Memory (henceforth, called CAM) that mirrors the composite action statements in the composite action and the causal links that is to be given to the next branch point. The causal links is nothing more than a set of causal links. However, CAM is slightly more complex. In essence, CAM is a tree structure in which the root corresponds to a composite action and the rest of the nodes correspond to a composite action statement in the composite action. Information regarding the state of the evaluation of each composite action statement is stored within its corresponding node in CAM. This allows the composite action state to use CAM for:

- Generating a new branch point for the current state.
- Getting all temporal constraints that should be applied after the current branch point. Exactly how this works is presented under temporal constraints in the next section.
- Querying if there is something left to do in the composite action.
- Getting all the variable bindings for the current scope.
- Generate labels for the current branch point. This will be covered under labels in the next section.

Composite action statement

Composite action statements (henceforth, simply called statements) are the building blocks of composite actions. In essence, they consist of ordered branch points and nested statements that TFPOP has to search through and evaluate respectively when using a composite action. In this report, the "if", "while" and "sequence" statements were implemented. Furthermore, the primitive action from the sequence action is also classified as a statement. The following sections cover these statements as well as some features that multiple statements can have. An overview of where the selected composite action components are implemented can be found in table 5.6.

| Composite action component | Implemented in |
|---|---|
| $(a_1; a_2; \dots; a_i)$ | The sequence statement |
| <i>with x_1, x_2, \dots, x_i where ϕ</i> | The variable introduction property |
| <i>if ϕ then a_1 else a_2</i> | The if statement |
| <i>while ϕ do a</i> | The while statement |
| $[t_0, t_1]$ | The labels and temporal constraint properties |

Table 5.6: A mapping from composite action component to where they are implemented.

Labels

In the sequence action, temporal constraints can be expressed by the index of the primitive action and with the keywords "#start" and "#end". However, this can be quite bothersome

to express for composite actions since they can include repeating structures (for example, iterations). Therefore, a labeling system was implemented that allows the invocation and effect nodes associated with a label to be accessed by "label#start" and "label#end" respectively. Those labels are added to a statement in TDDL is done by adding "#label" at the end of the statement. For example, the primitive action (*move ?uav ?thread ?base*) from the people-in-distress domain with the label *flyToBase* would be written as (*move ?uav ?thread ?base*)#*flyToBase*.

Another problem that arise due to the non-linear structure of the composite actions is that a label can occur multiple times. This is handled by attaching a suffix to each label. The suffix is generated by traversing CAMs tree structure where each node adds to the suffix according to table 5.7 until the node corresponding to the statement that has the label is reached.

| Memory object | Suffix |
|----------------------------|--|
| Composite action | "#" + the index of the current statement. |
| Primitive action statement | - |
| If statement | "->"; followed by "[extra=(" + all the extra variable bindings introduced in the statement + ")]" if there are any extra variable bindings were introduced; followed by "[?]" if the condition is not evaluated, "[false]" if the condition is <i>false</i> and "[true]" if the condition is <i>true</i> ; followed by "#" + the index of the current statement if the condition is evaluated. |
| While statement | "->"; followed by "[extra=(" + all the extra variable bindings introduced in the statement + ")]" if there are any extra variable bindings were introduced; followed by "[iter=" + the current iteration + "]" followed by "#cond" if the condition is the next branch point else "#" + the index of the current statement. |
| Sequence statement | "->"; followed by "[extra=(" + all the extra variable bindings introduced in the statement + ")]" if there are any extra variable bindings were introduced; followed by "#" + the index of the current statement. |

Table 5.7: Label suffix that is added at each memory node depending on what they are associated with.

There is one final problem that needs to be covered regarding the labels in composite actions, labels attached to non-primitive action statements. Time constraints applies to the nodes in a plan and therefore each label needs to be attached to a node. Hence, statements that do not directly add a node to the plan need to attach their labels to another statement's node instead. This is done by adding the start of a label to the first node that is added within its scope. The same principle applies for the end of a label except that it is added to the last node.

Temporal constraints

In composite actions it is not always efficient to apply time constraints after the whole action is finished since they can generate a long sequence of primitive actions. For example, there is one composite action that will solve the whole problem instance in the blocks world that is used in the search time experiments. If one imagine that there is a time constraint for each **move** that can never hold then the first **move** will be impossible and TFPOP will have to backtrack. However, if the time constraints are checked after the composite action is finished, then TFPOP will have to add all the primitive actions that the composite action generates

before the time constraint for the first fails. Therefore, it is preferable to be able to specify that time constraints should apply at a certain point in a composite action.

Based on the above reasoning it was decided to implement time constraints similarly to the introduction of new variable bindings. That is, as an optional property in the if, while and sequence statements as well as for the composite action as a whole. The syntax of time constraint in TDDL is the same as in the sequence action except that labels are used instead of indices. However, it can be written inside the statement or the composite action. Exactly when a time constraint is checked depends on where it is defined:

- **In an if statement:** Time constraints are tested when the last of the statements within the if statement has ended. It does not matter if the condition is *true* or *false*. However, the condition dictates which the last statement is.
- **In a while statement:** Time constraints are tested when each iteration is finished.
- **In a sequence statement:** Time constraints are tested when the last statement in the sequence statement is finished.
- **In a Composite action:** Time constraints are tested when the last statement in the composite action is finished.

Selecting which nodes belong to a time constraint is done by matching the labels. This is done by finding all the nodes that the composite action has added so far that have a label, ignoring suffix, that matches the start or the end label of the time constraint. Thereafter, each of the selected nodes has its labels distinguishing features, everything within brackets, extracted from the suffix. Then a search is done through the selected end nodes for each of the start nodes. The search tries to find a node with the same distinguishing features. If it finds one, then TFPOP will add the time constraint between those two nodes.

Introduce new variable bindings

Introduction of new variable bindings could be implemented as a separated statement on its own. However, it was decided to implement it as an optional property for the if, while and sequence statements since it reduces the needed number of statements. As a result, variables can be introduced in TDDL within the previously mentioned statements by using the ":with" and the optional ":where" keywords as shown in figure 5.4.

```

1 (:with (type-1 ?variable-1) (type-2 ?variable-2)...)
2 <(:where
3   ?variable-1 => condition-1
4   ?variable-2 => condition-2
5   ...
6 )>

```

Figure 5.4: The syntax for variable introduction in TDDL where everything within "<>" is optional.

During the search for a plan, the variable introduction is the first branch point in all statements that have one. This branch point is called **with where branch point** and belongs to the category of branch points that does not correspond to a primitive action. In the branch point, TFPOP chooses variable bindings satisfying the with clause and causal links satisfying the where clause (figure 5.5 shows the branches of this branch point). The selected variable bindings are added to in the scope of the statement and the causal links are passed to the next branch point in the scope. Note that these causal links will always be used in the scope since at least one primitive action or null action will be added in the scope. Finally, the conditions specified with the ":where" keyword are tested in the order in which they are written. As a result, the order they are defined can affect how fast the search for valid variable bindings and causal links are. When backtracking in this branch point, a new choice for variable bindings and/or causal links are chosen.

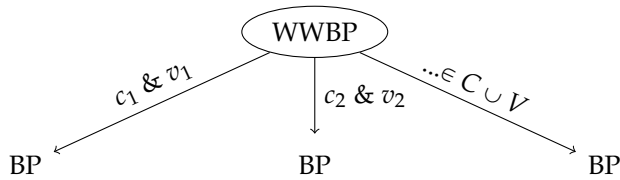


Figure 5.5: The different branches that a **with where branch point** has. WWBP stands for **with where branch point** and BP stands for a branch point. Furthermore, c_i stands for a set of causal links and v_i for a set of variable bindings. C is the set of all c_i and V is the set of all v_i .

If statement

The if statement introduces a way of adding different nodes to the plan depending on a boolean condition. This is a common term within programming and therefore the keywords for the if statement in TDDL is chosen to be similar to those that are commonly used (the syntax is shown in figure 5.6). However, there are two slight differences between the standard interpretation of an if statement and the one in TDDL. Firstly, instead of having a single condition, an if statement has a list of conditions where all of them have to be *true* or *false*. Secondly, a condition can be evaluated to both *true* and *false* depending on which causal links are added in some scenarios. The choice of how to make the conditions *true* or *false* corresponds to a new branch point, called **if branch point**, belonging to the category of branch points that do not correspond to a primitive action.

```

1 (:if
2   <(:with)>
3   <(:where)>
4   (:conditions non-empty-condition-list)
5   <(:eval-first truth value)>
6   (:then non-empty-statement-list)
7   <(:else non-empty-statement-list)>
8   <(:stnu)>
9 )
  
```

Figure 5.6: The syntax for an if statement in TDDL where everything within "<>" is optional.

The **if branch point** consists of two choices, one made by the one who writes the domain and one that TFPOP makes. Firstly, the choice made by the one who writes the domain is if TFPOP should try to make the conditions *true* first or *false* first. In TDDL, this is written with the optional option ":eval-first". However, if nothing is specified, then TFPOP will try to make the conditions *true* first. Secondly, the choice made by TFPOP is how to make the conditions *true* or *false*. Naturally, it starts with trying to find causal links that ensure that the conditions have the specified truth value. The selected causal links are the ones that are passed on to the next branch point within scope or added to the plan by using a null action. However, if no more causal links can be found that uphold the specified truth value, then TFPOP will try to make the conditions take to opposite value. When there is no more ways for TFPOP to make the conditions *true* or *false* it has to backtrack from the **if branch point**. The branches of the **if branch point** are illustrated in figure 5.7.

TFPOP applies an if statement by starting with a **with where branch point** if one exists. After that comes the **if branch point** and following that, it applies the statements for the selected truth value. All statements for *true* is specified with the ":then" keyword and the optional ":else" keyword for *false*. Note that both the ":then" and ":else" keywords require at least one statement. Moreover, the if statement is at the end if all the statements for the current truth value of the conditions has been applied. The whole evaluation procedure of the if statement is shown in fig 5.8.

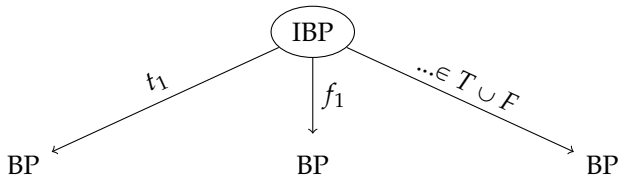


Figure 5.7: The branches that an **if branch point** has. IBP stands for **if branch point** and BP stands for a branch point. Furthermore, T is the set of all sets of causal links that make the conditions *true* and t_n is set number n in T . F and f_n have the same meaning as T and t_n except that the causal links make the conditions *false*.

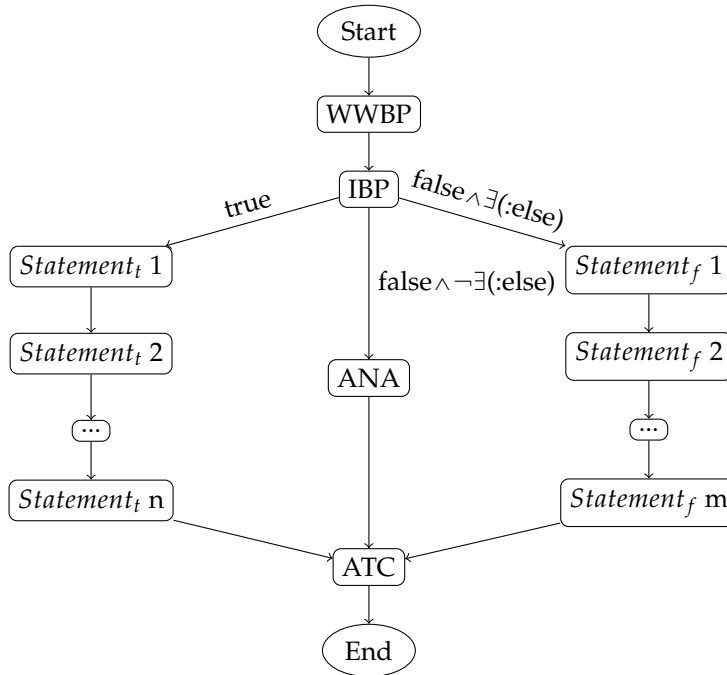


Figure 5.8: The evaluation procedure of an if statement. WWBP stands for **with where branch point** and IBP for **if branch point**. Furthermore, ATC stands for apply temporal constraints and ANA for add null action.

While statement

The while statement is a way of expressing that TFPOP should evaluate a sequence of statements zero or more times. Just like the if statement this is a common programming term. Therefore, the keywords of the while statement in TDDL are similar to the ones in many programming language (the syntax in TDDL is shown in figure 5.9). Moreover, the same differences regarding the conditions as in the if statements applies to the while statements. However, there are no statements for when the condition is *false* in the while statement. Therefore, it has a slightly different branch point for the conditions, called **while branch point**, that also belongs to the category of branch points that do not correspond to a primitive action.

TFPOP will try to find a set of causal links that makes the conditions *true* when in a **while branch point**. Moreover, when no previously tried sets can be found, then TFPOP will try to make the conditions *false*. Backtracking from a **while branch point** is done when all the possible set of causal links for both *true* and *false* has been tried. The branches of the **while branch point** are illustrated in figure 5.10.

```

1  (:while
2    <(:with)>
3    <(:where)>
4    (:conditions non-empty-condition-list)
5    (:body non-empty-statement-list)
6    <(:stnu)>
7  )

```

Figure 5.9: The syntax for a while statement in TDDL where everything within "<>" is optional.

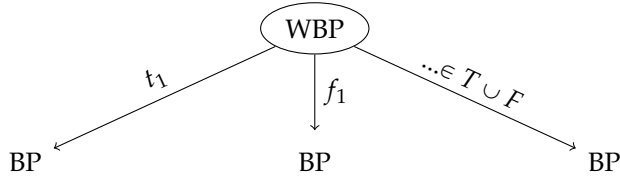


Figure 5.10: The different branches that a **while branch point** has. WBP stands for **while branch point** and BP stands for a branch point. Furthermore, T is the set of all sets of causal links that make the conditions *true* and t_n is set number n in T . F and f_n have the same meaning as T and t_n except that the causal links make the conditions *false*.

When evaluating a while statement, TFPOP starts with a **while branch point** and if the condition is evaluated to *true* then it will evaluate all the statements in the while statement. This is then repeated starting with new **while branch points** until one **while branch point** makes the conditions *false*. All of this is shown in figure 5.11.

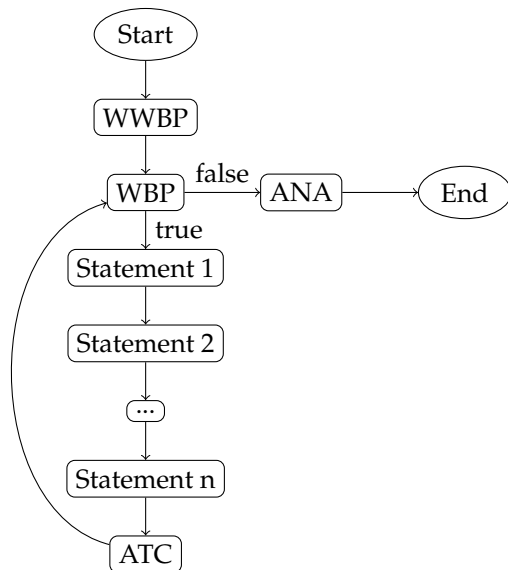


Figure 5.11: The evaluation procedure of a composite action while statement. WWBP stands for **with where branch point** and WBP for **while branch point**. Furthermore, ATC stands for apply temporal constraints and ANA for add null action.

Sequence statement

The sequence statement is a way of expressing a linear sequence of statements. This type of statements only differs from sequence actions in that it contains statements instead of primitive actions. The syntax for a sequence statement can be found in 5.12 and the evaluation model in figure 5.13.

```

1 (:sequence
2   <(:with)>
3   <(:where)>
4   (:body non-empty-statement-list)
5   <(:stnu)>
6 )

```

Figure 5.12: The syntax for a sequence statement in TDDL where everything within "<>" is optional.

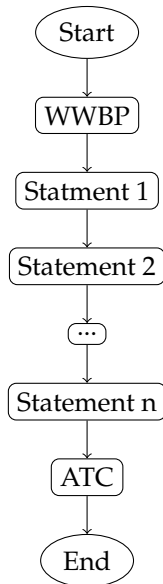


Figure 5.13: The evaluation procedure of a sequence statement. WWBP stands for **with where branch point** and ATC stands for **apply temporal constraints**.

Primitive action statement

The primitive action statement is the only branch point for the composite action that belongs to the category of branch points that add nodes to the plan. These statements find the needed causal links for the primitive action they correspond to and add the corresponding nodes to the plan. However, this branch point is just a renaming of the previously existing **sequence branch point** to **composite action branch point**. Moreover, the statement works just like the evaluation of one primitive action in the sequence statement. Therefore, it will not be covered any further.

Searching with composite actions

Composite actions are similar to the sequence action when it comes to how they are used by TFPOP during a search for a plan. Just like the sequence action, they can be selected by an **arbitrary action branch point** which takes care of binding the variables in the parameter list and creating causal links ensuring that the preconditions hold. After that, TFPOP evaluates all the statements in the composite action. Finally, all time constraints defined within the composite action is applied. This whole evaluation flow is shown in figure 5.14.

5.3 Search time

Compared to no guidance

The result from the experiment to test if composite actions can decrease the search time is summarized in table 5.8 and all data points are presented in appendix E. Moreover, applying Levene's test for equality of variance to the data gives that the variance of the two samples

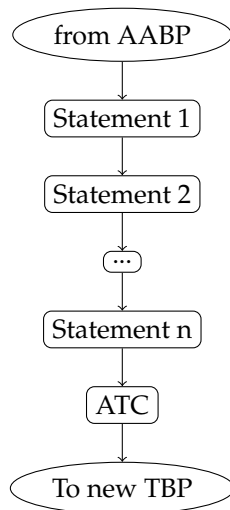


Figure 5.14: The evaluation procedure of a composite action. TBP stands for **thread branch point**, AABP stands for **arbitrary action branch point**, and ATC stands for **apply temporal constraints**.

are not equal ($F[19, 19]=31.33$, $p<.001$). Hence, a t-test that does not assume same variance is used to test for significance. Given the non-equal variance, TFPOP is solving the problem instance significantly faster (2 650.90 ms faster, 95% CI [2 615.53 ms, 2 686.27 ms]) when using composite actions to guide the search than when not using composite actions to guide the search ($t(19.00)=156.86$, $p<.001$, $d=49.60$).

| Domain type | Mean CPU time (ms) | Standard deviation |
|---------------------------|--------------------|--------------------|
| With composite actions | 1.4 | 0.5 |
| Without composite actions | 2652.3 | 75.58 |

Table 5.8: Mean and standard deviation for search time measured in milliseconds using the CPU for the different domains.

Compared to equivalent guidance

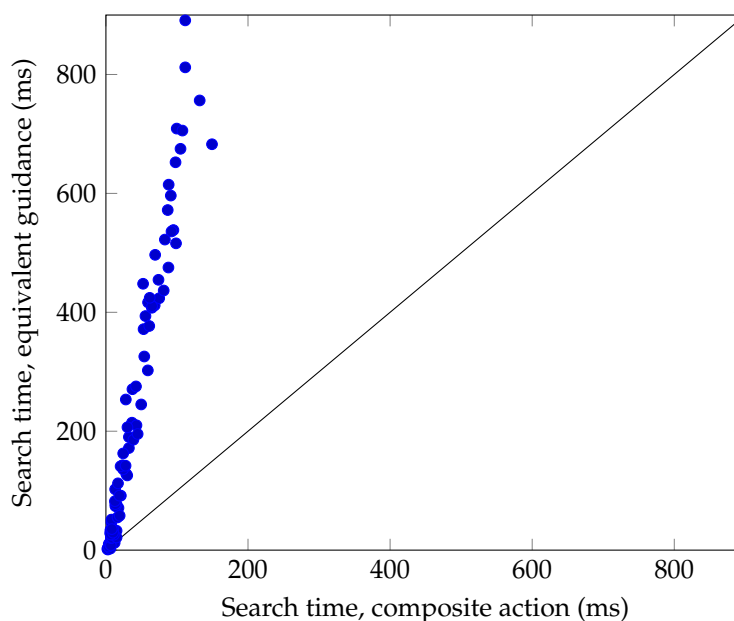


Figure 5.15: The search time for solving problem instances using different types of search guidance. Each dot is one problem instance. Furthermore, the x-axis is the search time when using guidance through composite actions and the y-axis is the search time when using equivalent guidance without composite actions. The line shows when both axes have the same value.

The search times it took to solve the randomized problem instances when guided by composite actions respectively with equivalent guidance without composite actions are shown in figure 5.15. Each dot in the graph represents one problem instance where the x value is the average time it took to create a plan when guided by composite actions and the y value is the average time it took to create a plan with equal guidance without composite actions. The line in the graph shows when the search time is equal for the two search guidance alternatives.

5.4 Understandability measurement

Throughout the search for understandability measurements there was only one that met the requirement. That was to measure understandability as the users' estimation of understandability in an experiment. Moreover, digging deeper in the field of measuring through asking questions with an answer scale led to measuring understandability through indirect questions.

5.5 Understandability experiment

Pre-experiment

There are two types of result from the pre-experiment. The first is an estimation of how large sample size is needed for a significant result and the second is the feedback from the participants and observations made during the experiment about the questionnaire. The following sections present these results.

All the data points from the pre-experiment can be found in appendix F. Moreover, the mean and standard deviation for *estimated understandability* and *measured understandability* is 1.1 and 1.08397 respectively 0.65 and 0.63369. Based on this, the estimated needed sample size for a significant result is 7 for both of the measurements.

The feedback and observations from the pre-experiment are that the experiment took slightly longer than 30 minutes and that some questions were hard to understand. Hence, all but one question were rewritten to make them easier to understand. The last question was removed from the questionnaire since it was the hardest to understand and would not have been possible to rewrite without introducing a whole new concept with examples. Moreover, the time measures were removed due to two reasons. Firstly, it did not give any information except how long time the experiment took. Secondly, it was observed that some participants seemed to act a bit stressed for the second domain in the questionnaire if the second took longer time than the first. As a result, the time questions were removed.

Experiment

It became quite obvious during the conduction of the experiment that there would not be enough participants to reach the needed sample size that was indicated by the pre-experiment. Therefore, the questionnaires were handed out to be done outside the experimental setting to increase the sample size according to the method (see section 4.5).

In total there were six participants (all males studying the five year computer engineering program at Linköping University). Five of the participants had previously encountered PDDL and the last stated a familiarity with the syntax through having knowledge about LISP. Moreover, three of the participants got the questionnaire where the PDDL file with a composite action was presented first. From the data (all data points are presented in appendix G), the following analysis was done using Wilcoxon signed-rank test and Pearson's r: The PDDL file with composite actions (median for *estimated understandability* is 1.5 and .875 for *measured understandability*) had a significantly higher *estimated understandability* ($z=-2.06$, $p=.020$, $r=-.60$) and a significantly higher *measured understandability* ($z=-2.20$, $p=.014$, $r=-.635$). In both cases, the effect was found to be large.



6 Discussion

6.1 Results

Composite action component ranking

The results for the composite action component ranking are not surprising. All the components except the introduction of new variables are standard concept in imperative programming. A composite action is an abstract action defined as an imperative procedure of primitive actions. Therefore, it is not surprising that the imperative concepts got a high rank. Moreover, the introduction of new variables is a powerful tool for allowing the planner to make partial decisions in the composite actions. Allowing for the user to express that it does not matter which the subject, place and/or object of the action is as long as it is done. Hence, the results are not unexpected.

One thing that should be mentioned is that BFM is based on Golog [1] which naturally creates a bias towards the components in Golog. However, this does not affect the decision of which components to implement since nearly all the composite action components that were chosen are present in all.

The fact that HTN supports recursion through named HLAs was unfortunately missed during this part of the report. As a result, recursion was ranked as two instead of three. This makes it one of the components that should have been implemented. However, the fact that recursion was not implemented does not affect the results for any of the experiments conducted in the report.

Implementation

There are some design decisions that are important to discuss and question. To begin with, the design decision regarding how the suffix of the labels are created and how they are used to matching the start and end of a time constraint is restricting the expressiveness of time constraints in TDDL. It was decided that the start and the end of a time constraint must have the same distinguishing features. This is a wanted feature so that the restrictions within, for example, a while-loop does not create any unintended time constraints between the iterations. However, this also makes it impossible to write a time constraint for the first statement in `:"then"` in an if statement and the statement following the if statement. This is because the if statement adds a new distinguishing feature ("`[true]`") when the condition is made *true*.

Hence, the two statements do not have the same distinguishing features. Therefore, the design is restricting which time constraints that can be expressed in TDDL. A better solution would have been to be more restrictive in which features are classified as distinguishing. Another approach would have been to let the user select how restricting the labeling system should be by using a flag.

The design decision to attach causal links to the nodes created by a phantom action if there are no nodes corresponding to a primitive action within the scope has two major implications. Firstly, this adds extra causal links to the plan which restricts the plan more. Secondly, this adds extra nodes to the plan. The first one is the worst since it might invalidate some plans that would otherwise be found. However, these extra causal links enforce the resulting plan to follow any composite actions that was used in it. Hence, the semantics of the composite actions become more similar to the one in most programming languages with this decision. The simplicity of the semantics is the reason for this design decision.

Search time

The result from the experiment regarding if composite actions can reduce the search time is not surprising. In essence, the composite action used in the experiment gives TFPOP the exact solution (through a procedure). Hence, only minimal search is required. On the other hand, when no search guidance is used, TFPOP has to search through the whole state space. This includes everything that is needed when using composite actions and much more. Therefore, this result was definitely expected. One should be aware that this is a specific case selected to show that it is possible to decrease the search time by using composite actions. Hence, no more general conclusions than that composite actions can decrease the search time can be drawn from this result.

It is quite easy to explain why guiding TFPOP with composite actions is faster than with an equivalent search guidance. The first thing to point out is that the equivalent search guidance contains more actions, more fluents and more objects. This results in that there is a larger state space. However, most of the combinations are not possible to select in a branch point since the preconditions prevent it. Nevertheless, there are still more branches to consider even if it is just to reject them. Moreover, the conditions in the while statement are a single branch point (if the conditions are *true*, otherwise three) in a composite action which means that TFPOP only has to expand one search node to finish it. However, the condition corresponds to two primitive actions (one for when the conditions are *true* and one for exiting the loop) in the equivalent guidance and one primitive action requires four branch points (see section 2.2). In summary, using composite actions lead to a smaller state space and fewer branch points. Given this it is natural that the guidance with composite actions is faster. However, all these reasons are dependent on the current implementation of TFPOP.

A more interesting thing about the results for the search time that compares search guidance through composite actions with equivalent search guidance is that the difference in search time is looking linear. Hence, this provides an indication that the search time for both of them are similar in time complexity. This indicates that the two search guidance methods are equally fast in theory. However, composite actions provides more expressiveness since it is, for example, possible to express time constraints over multiple actions.

Understandability measurement

One of the most important results regarding the understandability measurement is that no metric that is applicable and is well cited was found. There were a lot of metrics that were studied throughout the report. However, these can be categorized into three different categories. Those that are well cited but not applicable (for example, QMOOD), those that are applicable but not validated for understandability (for example, code and data spatial

complexity) and those that are not well cited (for example, improved cognitive information complexity). Unfortunately, non of these categories are usable for measuring TDDL.

The metrics that are not applicable make use of some properties that are not present in TDDL. QMOOD is based on properties of object oriented design [3]. This includes properties like encapsulation which do not have any corresponding property in TDDL. Hence, it is not possible to adapt the metric without creating an issue with validity.

Applicable metrics are, on the other hand, based on properties that exist in TDDL. These properties are, for example, data definition and lines of code. Improved cognitive information complexity is an example that is applicable. For example, by counting properties in a TDDL file as variables, boolean operators as operators and variables as variables it is possible to calculate the cost for each line and thereby the whole metric. However, no metric that was applicable was found to have enough citations to be considered acceptable.

There are some metrics that have been proposed that are not validated properly. A prime example of this is the code and data spatial complexity. This metric is validated through its correlation with maintenance time [6]. However, there is no proof or arguments that understandability is the only factor among the different treatments in the experiment that affects the maintainability [6]. Hence, the experiment is actually validating the metric as a metric measuring maintainability.

Understandability measurements that can be used in experiments proved to give better results than the metrics. However, these included some that were not possible to use in this report. To begin with, all the measurements that needed experts were unfortunately outside the scope of the resources for the report. Moreover, there were some measurements that are not validated, just like for the metrics. For example, using time and correct answers on knowledge questions is one measurement that was used [9]. However, this measurement is never validated as a measurement of understandability in the study using it [9]. Therefore, this is not an applicable measurement. This leaves two possible measurements: user estimation and indirect questions.

One major drawback with the indirect questions is that it needs to be validated that they measure understandability. As a result, this measurement can not be used to draw conclusions in this experiment since no validated set of questions was found. However, it can be used to get an indication. Moreover, using the aspects of understandability that Boehm, Brown and Lipow identified [4] as a foundation for the indirect questions increases the credibility of such an indication.

Understandability experiment

The results showed that the understandability is significantly higher when using composite actions. From this there are two interesting parts to notice. The first is that it is an indication that the indirect questions managed to measure the estimated understandability and the second is regarding why the result is as it is. Naturally, there are multiple reasons to why the composite actions have a significantly higher understandability. Firstly, the model that uses composite actions is smaller and if assuming that the principle that larger designs are harder to understand [3] applies to a domain model then this helps to explain the result. Secondly, the participants have been using basic control structures more often than they have been using PDDL. Hence, the familiarity might explain parts of the result. Finally, composite actions give a direct description of an abstract action while defining extra primitive actions with extra preconditions give an indirect description. In addition, a direct description is most likely easier to understand. Therefore, this also helps to explain the result.

6.2 Method

Composite action components ranking

As written in the delimitations (section 1.4), this study will not research which is the most important composite actions. Therefore, the only threat to the reliability of the method is the categories that were identified for the composite action components. Unfortunately, there is no method for how the categorization is done and therefore the reliability of this part can be considered low. However, this does not affect the report as a whole since it does not affect the validity or reliability of the main research questions.

Implementation

The implementation that was done did not follow any specific method or algorithms. Moreover, the source code is not published and therefore every measure based on TFPOP is not replicable with a high reliability. However, it is possible to replicate since the principles that TFPOP is built on are published by Kvarnström [14] and in this report. However, this implies that the reliability is lowered since it is most likely that there will be differences in the implementation.

Search time

The method chosen for determining if using composite actions can decrease the search time is a statistical analysis of samples for proving differences. Hence, it is valid to conclude that composite actions can decrease the search time in case a significantly (in the statistical meaning) faster result is found. However, "can" is an operative word since no valid conclusions about a general case can be drawn because only one case is analyzed. The test is deterministic and therefore it is also reliable. However, the source code is not publicly available and therefore it is hard to replicate the experiment (see section 6.2).

The experiment used to get an indication of how well search guidance with composite actions compare to equivalent search guidance without composite actions inherits the problem with replicability from the implementation. In addition, the method used for generating the samples is randomized which makes it possible to get another result. Moreover, the method to analyze the result does not take this into consideration. Therefore, this experiment has high problems with validity if a conclusion would be drawn from it. Nevertheless, the aim is only to get an indication. Hence, that should not be an issue for validity.

Understandability measurement

The method chosen for this part of the report has two major flaws regarding validity. Firstly, citation is used to measure how established a measurement is. This is a flaw since it is assuming that citations are positive. However, this is not always correct. In fact, the opposite can be true. Secondly, the method is limited to some keyword for searching for measurements. These two flaws potentially exclude measurements that are better than the ones that was found. These two issues show that it is not valid to draw the conclusion about the best measurement method. However, they do not invalidate the results as a measurement for understandability.

One more issue with the method is that the reliability is decreasing with time since the basis for the search is *www.scholar.google.com* which constantly gets new publications and the number of citations constantly changes. As a result, it gets harder to get the same result with a replicated search for measurement with time even if the limit is increased.

Understandability experiment

There is one error source in the method regarding controlling the variables in the experiment. This problem occurred when questionnaires were handed out to the participants to fill in outside the experimental setup. As a result, there is no guarantee that the participants, for example, did not take any breaks between the two parts of the questionnaire. Therefore, there might be other causes for the result than using composite actions or not. However, all the participants signed a consent form that includes that they had understood the instructions for the experiment. Hence, it is quite safe to assume that this does not prove a huge issue for the validity.

Using the users estimated understandability as a measurement has one major effect on any conclusions. The effect is that the estimated understandability is a subjective measurement. This does actually have two major implications. The first is that each participant may have their own interpretation of the meaning of the alternatives on the answering scale. Hence, making it hard to infer anything between the domains for each subjects. However, this is taken care of with the within-subject design which only measures the difference between each subject. The second problem with the estimated understandability is that it can not be used to draw any conclusions about an objective understandability. Hence, the conclusions can only be drawn about the subjective (or felt) understandability.

The fact that the population that the sample was randomized from is students at Linköping University with at least basic knowledge about planning affects the conclusions that can be drawn. However, Kitchenham et al. state that this is not a hinder when it comes to drawing a conclusion for non-experts [11]. Therefore, this restriction applies to all the conclusions that are drawn from this experiment. However, this means that if Kitchenham et al. is wrong then the all conclusions has to be restricted further so that they only apply to the population of the sample. Disregarding if Kitchenham et al. is correct or wrong, a significant result can be seen as an indication of that the understandability is higher for experts as well.

There are some problems with the questionnaire that was used in the experiment. The problems originate from the need of conducting the experiment in time. As a result, the questionnaire and updating the questionnaire was not given enough time. Therefore, it contains some language errors in the description. Not large enough to prevent the participants from understanding but enough to require more effort. Moreover, there are two places where the participant is asked to write down the current time which should have been removed. However, no place to write down the time is presented. Nevertheless, non of the found language errors are present in the questions or the domain file. Hence, this should not prove any problem to the validity of the experiment. Moreover, no participant wrote down the time in any of the places they were asked to. Hence, it is believable that they either missed it or ignored it. Therefore, this should not be a problem either. As a result, the errors in the questionnaire should not affect the result.

One final question about the experiment to discuss is regarding how applicable a result for PDDL is on TDDL. To begin with, both TDDL and PDDL are based on LISP and therefore have a strong resemblance with each other. This is a good foundation for claiming that the results for the experiment can be used to conclude if composite actions can improve the understandability of TDDL. Furthermore, it is only the difference between using composite actions and not using them that is measured. Moreover, the composite actions used for PDDL are designed to be similar to those of TDDL. Hence, the difference between using composite actions or not should be similar between PDDL and TDDL. These two arguments make a strong claim that valid conclusions can be drawn for TDDL as well as for PDDL.

6.3 Source criticism

There are two main things to consider regarding the sources used in the report. Firstly, most of the sources regarding composite actions are around 15 years old. However, this does not

affect the report since these are only used for generating an informal ranking of what to implement and not as basis for any conclusions. Secondly, a few of the cited publications regarding metrics should be used with caution. The reason for this is that there are multiple publications that criticize them. Disregarding of the validity of the criticism, these publications are only used as examples of metrics that is not going to be used. Therefore, this does not affect the results.

6.4 The work in a wider context

The report aims to extend modeling languages for planners so that parts of the domain become easier to understand. Making them easier to understand is also likely making them easier to write. As a result, this can affect how easy it is to use a planner, regardless what they are used for. This includes applications like the people in distress example and the repairs of a nuclear reactor from the introduction. Unfortunately, this also includes creating autonomous machines for war. In essence, the result from this report is one step towards making it easier, but not necessarily more effective, to use planning autonomous systems regardless of their purpose.



7 Conclusion

7.1 Research questions

From the results in this study the following conclusions can be drawn regarding the research questions:

1. The implementation proves that composite action components can be implemented in TFPOP.
2. Composite actions can be used to guide TFPOP when searching for a plan and thereby, decreasing the search time. Moreover, there are indications that the composite actions provide search guidance that is as effective as the equivalent search guidance that does not use composite actions.
3. Using guidance in the form of composite actions makes a domain written in TDDL have a higher felt understandability for non-expert users than if equivalent guidance was used in the domain. Moreover, the results indicate that using composite actions increase the felt understandability for experts compared to using equivalent guidance.

7.2 Further work

Implementation

There are still a lot of work that can be done regarding the implementation when it comes to composite actions in TFPOP. The most obvious is to implement the rest of the composite action components identified in this report. A more interesting study is to research the possibility for the planner to learn and use commonly used composite actions for a problem instance. Moreover, this can be extended to learn composite actions that are useful for all problem instances and domains with a set of properties. Finally, it would prove beneficial in terms of not restricting the resulting plan if it is possible to always attach the causal links that are enforced by all composite action components at the latest moment they are needed instead of naively attaching them to the next node that is added to the plan.

Understandability measurement

The main issue that can be worked with when it comes to understandability measures for languages like PDDL (declarative languages in general) is the lack of metrics. A metric allows automatic code reviews of abstract concepts during development. For example, a metric for understandability could be used to give warnings when a piece of code is considered unnecessarily hard to understand. Combining this with an expert system that gives hints about how to improve the understandability could potentially increase the understandability (and thereby, the maintainability) during the development process. Therefore, the first step that needs to be taken regarding understandability measurements for this kind of languages is to develop and validate a metric.



A Metric formulas

A.1 Code and data spatial complexity

The code-spatial complexity metric is calculated with the following formula [6]:

$$\frac{\sum_{i=0}^m \frac{\sum_{j=0}^{n_i} d_{i,j}}{n_i}}{m}$$

Where m is the number of modules in the software, n_i is the number of uses of module i and $d_{i,j}$ is the distance, measured in lines of code, from the definition of module i to call j to module i . When the module is in another file, the distance is defined to be the sum of the distance from the call to the bottom of that file and the distance from the top of the file that the module is defined in to the definition of the module [6].

The data-spatial complexity is calculated as follows:

$$\frac{\sum_{i=0}^q \frac{\sum_{j=0}^p d_{i,j}}{p}}{q}$$

Where q is the number of global data members, p is the number of times that data member is used and $d_{i,j}$ is the distance to the previous use of the data member, in lines of code, or the definition if there is no previous use.

A.2 Improved cognitive information complexity

The improved cognitive information complexity metric starts with calculating the information content of a line with [13]:

$$I_n = 4 * Identifiers_{(b,n)} + Identifiers_{(g,n)} + Operators_n$$

Where n is the line number and $Identifiers_{(b,n)}$ is the count of all the identifiers with bad names (with regard to the domain of the software) on line n . Furthermore, $Identifiers_{(g,n)}$ is the count of all the identifiers with good names on line n and $Operators_n$ is the sum of all the operators on line n .

The information content is then used to calculate how much the information will impact on the following lines according to:

$$WICL_n = I_n / (L - n)$$

Where L is the number of lines of code in the software. The sum of the impact of all the lines of code is the weighted information count of the software ($WICS$).

The next step is to calculate the cognitive weights of the basic control structures ($WBCS$) as presented by Wang and Shao [20]:

$$WBCS_n = C(BCS_n) * \sum_{i=0}^m WBCS_i$$

Where C is a function that returns the cost for a basic control structure and the summation iterates over all the directly nested basic control structures within BCS_n . Furthermore, $WBCS_n$ is the weighted cost of the basic control structure BCS_n . From this the sum of all basic control structures ($SWBCS$) in a piece of software is calculated as [20]:

$$SWBCS = \sum_{i=0}^n WBCS_i$$

Where the summation iterates over all the non-nested basic control structures.

The final piece of the formula for basic control structure is the function C which simply maps a basic control structure to a value according to table A.1 [20].

| Control structure | Cost |
|----------------------------------|------|
| Sequence | 1 |
| If-then(-else) | 2 |
| Case (Switch) | 3 |
| For-loop | 3 |
| Do-While | 3 |
| While | 3 |
| Recursion | 3 |
| Function call | 2 |
| Parallel (e.g. multiple threads) | 4 |
| Interrupt | 4 |

Table A.1: The cost for different basic control structures according to Wang and Shao.

The final part of calculating the cognitive information complexity is to combine the cost for the basic control structures and the weighted information count [13]:

$$CIC = WICL * SWBCS$$

B Blocks World

```
1 (:tfpop-domain blocks-world
2   (:flags)
3   (:types (object (block) (ground)) (agent))
4   (:constants (ground floor))
5   (:fluents
6     (object (on-top block))
7     (boolean (clear object))
8   )
9
10  (:operator (move (agent ?a) (thread ?thread) (object ?from) (block ?b) (object ?to))
11    (:split-precond
12      ?thread => (:true)
13      ?from => (:true)
14      ?b => (and (= (on-top ?b) ?from) (= (clear ?b) true) (not (= ?from ?b)))
15      ?to => (and (or (= (clear ?to) true) (= ?to floor)) (not (= ?from ?to)))
16    )
17    (:phase
18      (:duration 1 1 1)
19      (:effects
20        (:= (on-top ?b) ?to)
21        (:= (clear ?from) true)
22      )
23    )
24    (:always-followed-by)
25    (:definitely-changes (on-top ?b))
26    :can-achieve-goal
27  )
28 )
```

Figure B.1: Blocks world domain without any extra guidance.

```

1 (:tfpop-domain blocks-world
2   (:flags)
3   (:types (object (block) (ground)) (agent))
4   (:constants (ground floor))
5   (:fluents
6     (object (on-top block))
7     (boolean (clear object))
8   )
9
10  (:operator :onlypart (move (agent ?a) (thread ?thread) (object ?from) (block ?b) (object ?to))
11    (:split-precond
12      ?thread => (:true)
13      ?from => (:true)
14      ?b => (and (= (on-top ?b) ?from) (= (clear ?b) true) (not (= ?from ?b)))
15      ?to => (and (or (= (clear ?to) true) (= ?to floor)) (not (= ?from ?to)))
16    )
17    (:phase
18      (:duration 1 1 1)
19      (:effects
20        (:= (on-top ?b) ?to)
21        (:= (clear ?from) true)
22      )
23    )
24    (:always-followed-by)
25    (:definitely-changes (on-top ?b))
26    :can-achieve-goal
27  )
28
29  ;; Flattens all towers by setting the blocks on the floor
30  (:composite-operator (flatten-towers (agent ?a) (thread ?thread) (block ?firstBlock)
31    (block ?firstFrom))
32    (:split-precond
33      ?thread => (:true)
34      ?firstBlock => (= (clear ?firstBlock) true)
35      ?firstFrom => (= (on-top ?firstBlock) ?firstFrom)
36    )
37    (:body
38      (move ?a ?thread ?firstFrom ?firstBlock floor)
39      (:while
40        (:conditions (exists (block ?b) (not (= (on-top ?b) floor))))
41        (:body
42          (:sequence
43            (:with (block ?nextBlock) (block ?nextFrom))
44            (:where
45              ?nextBlock => (= (clear ?nextBlock) true)
46              ?nextFrom => (= (on-top ?nextBlock) ?nextFrom)
47            )
48            (:body
49              (move ?a ?thread ?nextFrom ?nextBlock floor)
50            )
51          )
52        )
53      )
54    )
55  )
56 )

```

Figure B.2: Blocks world domain with composite action as extra guidance.

```

1 (:tfpop-domain blocks-world
2   (:flags)
3   (:types (object (block) (ground)) (agent) (op))
4   (:constants (ground floor) (op FT-cond) (op FT-while-body))
5   (:fluents
6     (object (on-top block))
7     (boolean (clear object))
8     (boolean (is-current-agent agent))
9     (boolean (is-current-op op))
10  )
11
12  (:operator :onlypart (move (agent ?a) (thread ?thread) (object ?from) (block ?b) (object ?to))
13    (:split-precond
14      ?thread => (:true)
15      ?from => (:true)
16      ?b => (and (= (on-top ?b) ?from) (= (clear ?b) true) (not (= ?from ?b)))
17      ?to => (and (or (= (clear ?to) true) (= ?to floor)) (not (= ?from ?to)))
18    )
19    (:phase
20      (:duration 1 1 1)
21      (:effects
22        (:= (on-top ?b) ?to)
23        (:= (clear ?from) true)
24      )
25    )
26    (:always-followed-by)
27    (:definitely-changes (on-top ?b))
28    :can-achieve-goal
29  )
30
31  ;; a composite operator expressed as standard operators
32  (:operator (flatten-towers-init (agent ?a) (thread ?thread) (block ?firstBlock) (block ?firstFrom))
33    (:split-precond
34      ?thread => (:true)
35      ?firstBlock => (= (clear ?firstBlock) true)
36      ?firstFrom => (= (on-top ?firstBlock) ?firstFrom)
37    )
38    (:phase
39      (:duration 1 1 1)
40      (:effects
41        (:= (on-top ?firstBlock) floor)
42        (:= (clear ?firstFrom) true)
43        (:= (is-current-op FT-cond) true)
44        (:= (is-current-agent ?a) true)
45      )
46    )
47    (:always-followed-by flatten-towers-while-condition-true flatten-towers-while-condition-false)
48    (:definitely-changes (on-top ?firstBlock) (clear ?firstFrom) (is-current-op FT-cond))
49    :can-achieve-goal
50  )
51
52  (:operator (flatten-towers-while-condition-true (agent ?a) (thread ?thread) (block ?b))
53    (:split-precond
54      ?thread => (and (= (is-current-agent ?a) true) (= (is-current-op FT-cond) true))
55      ?b => (and (= (clear ?b) true) (exists (block ?n) (not (= (on-top ?b) floor))))
56    )
57    (:phase
58      (:duration 1 1 1)
59      (:effects
60        (:= (is-current-op FT-while-body) true)
61        (:= (is-current-op FT-cond) false)
62      )
63    )
64    (:always-followed-by flatten-towers-while-body)
65    (:definitely-changes (is-current-op FT-while-body) (is-current-op FT-cond))
66  )

```

Figure B.3: Blocks world domain with guidance equivalent to the composite action in B.2, part 1.

```

68  (:operator (flatten--towers--while--condition--false (agent ?a) (thread ?thread) (block ?b))
69    (:split-precond
70      ?thread => (and (= (is-current-agent ?a) true) (= (is-current-op FT-cond) true))
71      ?b => (and (= (clear ?b) true) (not (exists (block ?n) (not (= (on-top ?b) floor)))))
72    )
73    (:phase
74      (:duration 1 1 1)
75      (:effects
76        (:= (is-current-op FT-cond) false)
77        (:= (is-current-agent ?a) false)
78      )
79    )
80    (:always-followed-by)
81    (:definitely-changes (is-current-op FT-cond) (is-current-agent ?a))
82  )
83
84  (:operator (flatten--towers--while--body (agent ?a) (thread ?thread) (block ?b) (block ?from))
85    (:split-precond
86      ?thread => (and (= (is-current-agent ?a) true) (= (is-current-op FT-while-body) true))
87      ?b => (= (clear ?b) true)
88      ?from => (= (on-top ?b) ?from)
89    )
90    (:phase
91      (:duration 1 1 1)
92      (:effects
93        (:= (on-top ?b) floor)
94        (:= (clear ?from) true)
95        (:= (is-current-op FT-cond) true)
96        (:= (is-current-op FT-while-body) false)
97      )
98    )
99    (:always-followed-by flatten--towers--while--condition--true flatten--towers--while--condition--false)
100   (:definitely-changes (on-top ?b) (clear ?from) (is-current-op FT-cond)
101     (is-current-op FT-while-body))
102   :can-achieve-goal
103 )
104 )

```

Figure B.4: Blocks world domain with guidance equivalent to the composite action in B.2, part 2.

```

1  (:tfpop-problem blocks-1
2    (:domain blocks-world)
3    (:agents (agent (arm arm-thread)))
4
5    (:objects (block A B C D E F))
6
7    (:init
8      (= (on-top A) floor)
9      (= (on-top B) A)
10     (= (on-top C) B)
11     (= (on-top D) C)
12     (= (on-top E) floor)
13     (= (on-top F) floor)
14     (= (clear A) false)
15     (= (clear B) false)
16     (= (clear C) false)
17     (= (clear D) true)
18     (= (clear E) true)
19     (= (clear F) true)
20     (= (clear floor) false)
21   )
22   (:goal
23     (= (on-top A) floor)
24     (= (on-top B) floor)
25     (= (on-top C) floor)
26     (= (on-top D) floor)
27     (= (on-top E) floor)
28     (= (on-top F) floor)
29   )
30 )

```

Figure B.5: Blocks world problem used to compare search time when comparing to no guidance.

```

1 (:tfpop-problem generated-blocks-problem-4-ca
2   (:domain blocks-world)
3   (:agents (agent (arm arm-thread))))
4   (:objects (block A B C D E))
5   (:init
6     (= (on-top A) floor)
7     (= (on-top B) A)
8     (= (on-top C) floor)
9     (= (on-top D) B)
10    (= (on-top E) floor)
11    (= (clear floor) false)
12    (= (clear A) false)
13    (= (clear B) false)
14    (= (clear C) true)
15    (= (clear D) true)
16    (= (clear E) true)
17  )
18  (:goal
19    (= (on-top A) floor)
20    (= (on-top B) floor)
21    (= (on-top C) floor)
22    (= (on-top D) floor)
23    (= (on-top E) floor)
24  )
25 )

```

Figure B.6: Example problem used for domain with composite action when comparing with domain using equivalent guidance.

```

1 (:tfpop-problem generated-blocks-problem-4-equal
2   (:domain blocks-world)
3   (:agents (agent (arm arm-thread))))
4   (:objects (block A B C D E))
5   (:init
6     (= (on-top A) floor)
7     (= (on-top B) A)
8     (= (on-top C) floor)
9     (= (on-top D) B)
10    (= (on-top E) floor)
11    (= (clear floor) false)
12    (= (clear A) false)
13    (= (clear B) false)
14    (= (clear C) true)
15    (= (clear D) true)
16    (= (clear E) true)
17    (:all is-current-op false)
18    (:all is-current-agent false)
19  )
20  (:goal
21    (= (on-top A) floor)
22    (= (on-top B) floor)
23    (= (on-top C) floor)
24    (= (on-top D) floor)
25    (= (on-top E) floor)
26  )
27 )

```

Figure B.7: Example problem used for domain with composite action when comparing with domain using equivalent guidance.



Questionnaire

Swedish version used in experiment

Förståbarhet hos PDDL

Experimentets utformning

I experimentet kommer du först att bli ombedd att fylla i lite bakgrundsinformation. Därefter får du läsa en kort introduktion till sammansatta handlingar, en utlösning till PDDL. Utöver introduktionen får du även en kort beskrivning över domänen som används i experimentet samt en graf som beskriver flödet i en sammansatt handling som kommer användas senare i experimentet. Ryck gårna ut sidan med introduktionen och grafen och den vid sidan av till resten av experimentet. Efter de ska du svara på ytterligare PDDL-fil och svara på frågorna om den. När du är färdig ska du svara på ytterligare några frågor. Processen upprepas därefter för ytterligare en utskrivet PDDL-fil.

Medgivande

Jag ställer upp på att vara med på studien ledd av Erik Hansson för Linköpings Universitet.

Jag är fullt medveten om att det är frivilligt att delta i studien och att jag när som helst kan avbryta sessionen om jag känner mig obekväm med den. All data som samlas in kommer vara anonym förutom för deltagarna i forskningsgruppen. Resultatet av studien är tillgängligt när rapporten publiceras. Alla resultatdeltagare har rätten att få ut data för sitt egna deltagande.

Vänligen skriv under på att du har tagit del av och förstått ovanstående information samt att du har fått svar på alla frågor om sessionen.

Datum: _____

Underskrift: _____

Namn/fortydligande: _____

Tack för att du deltar.

Kontaktinformation:
Erik Hansson
ehh1a172@student.liu.se

Bakgrundsfrågor

1. Jag är:
 Man Kvinna
2. Vilket program studerar du vid? _____
3. Vilken temin är du registrerad på? _____
4. Har du tidigare varit i kontakt med området planering inom artificiell intelligens?
 Ja Nej
5. Har du använt PDDL eller liknande språk tidigare?
 Ja Nej

Introduktion till sammansatta handlingar

Sammansatta handlingar (en. composite actions) är ett sätt att uttrycka ett block av handlingar (en. actions) och av andra block av handlingar som planeraren kan använda tillsammans. Utformningen de har är lik den i vanliga programmeringsspråk. Det kan till exempel vara en if-sats, while-loop, en sekvens eller en ensam handling. Utformningen på en sammansatt handling i PDDL liknar den vanliga handlingen och har formen:

```
(:compositeAction name
  :parameters (list of variables)
  :precondition (logical expression)
  :body (description of the
  :block) (logical expression))
```

De nya elementen är "body" vilket innehåller ett block av handlingar och andra block av handlingar samt "pledge" vilket är ett logiskt uttryck som beskriver vad den sammansatta handlingen lovar ska vara sant efter att handlingen har utförts. Notera att det inte är hela effekten utan mer kan påverkas av den sammansatta handlingen.

Ett block av typen *typen* uttrycks i sin tur på formen:

```
(:type
 <:with (list of variables)>
 <:where (logical expression)>
 type specific inputs)
```

Här är "with" ett icke-deterministiskt val av parametrar som planeraren gör varje gång blocket utförs och "where" är ett logiskt uttryck som säger vad som måste vara sant för parametrarna i "with". De båda är frivilliga att ha med men det finns en del typspecifika delar som måste vara med. Till exempel måste typen "sequence" (en linjär sekvens av handlingar och block) och typen "while" ha "body" som listar vilka block och handlingar den består av. Vidare måste typerna "if" och "while" ha "condition" som beskriver vad som testas. "if" måste även ha "then" som är en lista över block och handlingar som ska utföras om "condition" är sann. Vidare kan "if" ha "else" som specificera de block och handlingar som görs om condition är falsk. En speciell typ är "singleAction" som tillåter att "with" och "where" specificeras för en enda handling. Handlingen i färg står under "do".

Avancerade predikat

För att abstrahera logiska uttryck och få en högre uttryckskraft används utöver de sammansatta handlingarna flera mer avancerade predikat än de vanliga predikaten i PDDL. De är beskrivna på följande form:

```
(:definition:Predicate name
  :parameters (list of variables)
  :predicate (logical expression))
```

Likt de vanliga predikaten kan de här användas i logiska uttryck med namnet och parametrarna till predikatet. En fördel med de här är att de kan uttrycka sig rekursivt e.g. (`:predicate (and (isTrue ?x) (name ?y))`). Utöver det fungerar de exakt som vanliga predikat som testas genom (`name ?x1 ?x2 ... ?xn`).

goal-funktionen

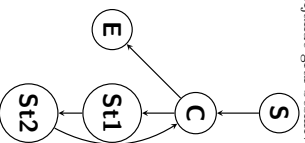
När man skriver en domän-fil är det inte ovanligt att man vill vägleda planeraren genom att tillåta domänfilen att referera till målet i problemdefinitionen. Det är användbart

eftersom det är onödigt att utföra en handling om den varken kan ge nya möjligheter eller om den inte leder till något som är ett mål. För att specificera de sakerna brukar en goal-funktion användas. Den tar en parameter som är ett logiskt uttryck och testar om det logiska uttrycket finns med bland målen i det aktuella problemet. Till exempel blir predikatet *goal (isTrue ?x)* enkelt sagt om *(isTrue ?x)* är med bland målen i problemet när man har ersatt *?x* med värdet det har för närvarande.

PDDL domänen

I experimentet kommer en och samma domän användas för båda PDDL-filerna. Domänen är blocks-world vilken består av block, en robotarm och marken. Robotarmen kan användas för att plocka upp block som inte har något annat block på sig samt för att sätta ner blocken den håller i. Problemen i blocks-world består av att bygga ett eller flera torn utifrån ett eller flera torn.

För att hjälpa till att vägleda planeraren har en sammansatt handling, *clearBlock*, lagts till. Handlingen frigör ett block oavsett hur många block som finns ovanpå det. Följande graf beskriver *clearBlocks* kontroll flöde:



Första PDDL filen

På kommande sidor kommer du få se en PDDL fil och på sidan därefter finns det 4 frågor angående PDDL filen. Svara på frågorna så exakt du kan. Det är helt fritt att bläddra tillbaka till PDDL filen på nästa sida när du besvarar frågorna. Glöm inte att anteckna den aktuella tiden innan du börjar.

```

1  ;; This is a domain file modeling the block-world where a robot arm stacks
2  ;; blocks on the ground and on each other so that desirable towers are created.
3
4  (define (domain block-world)
5    (:requirements :strips :adl :equality)
6
7    (:constants GROUND ARM)
8
9    ;; All predicates are modeling predicates
10   (:predicates (on ?x ?y) (clear ?x) (block ?x) (holding ?x) (empty ?x))
11
12   ;; Internal predicate that checks if ?top is the block that is on top of
13   (:definePredicate topOfStack
14     parameters (?object ?top)
15     :predicate (and (block ?object) (block ?top)
16                   (or (on ?top ?object)
17                       (exists (and (block ?middle)
18                                   (on ?middle ?object)
19                                   (topOfStack ?middle ?top))
20                           )))
21
22   )
23
24   )
25
26   ;; Primitive actions ;;
27   (:action pickUp
28     parameters (?b ?om)
29     :precondition (and (block ?b) (or (block ?om) (= ?om GROUND))
30                          (clear ?b) (on ?b ?om) (empty ARM))
31     :effect (and (holding ?b) (clear ?om) (not (on ?b ?om))
32                  (not (empty ARM)))
33   )
34
35   (:action putDown
36     parameters (?b ?om)
37     :precondition (and (block ?b) (or (and (block ?om) (clear ?om))
38                          (= ?om GROUND))
39                          (holding ?b) (not (empty ARM)))
40     :effect (and (not (holding ?b)) (on ?b ?om) (not (clear ?om))
41                  (empty ARM))
42   )
43
44   ;; Composite actions ;;
45
46   ; Clear a block that is supposed to be clear according to the goal but
47   ; isn't at the moment. Puts all the blocks on top on the ground
48   (:compositeAction clearBlock
49     parameters (?b)
50     :precondition (and (block ?b) (not (clear ?b))) (empty ARM)
51     :body (while (goal (clear ?b)))
52            :body (sequence
53                  :condition (not (clear ?b))
54                  :with (?orTop ?secondFromTop)
55                  :where (and (block ?orTop)
56                              (block ?secondFromTop)
57                              (topOfStack ?orTop ?b)
58                              (topOfStack ?secondFromTop ?orTop))
59                  :body ((pledge (on ?orTop ?secondFromTop ?b)
60                                (putDown ?orTop GROUND)))
61                )
62            :pledge (clear ?b)
63          )
64
65
66
67

```

Frågor på första PDDL filen

1. Hur många handlingar finns det totalt delmarkerade?

2. Hur många handlingar påverkar minst ett modellerande predikat (se kommentarer för predikat i PDDL-filen för vilka predikat som är modellerande)?

| | | | | | |
|---|--------------------------|---|--------------------------|---|--------------------------|
| 1 | <input type="checkbox"/> | 2 | <input type="checkbox"/> | 3 | <input type="checkbox"/> |
|---|--------------------------|---|--------------------------|---|--------------------------|
3. Vilka handlingar (en. actions) är möjliga (ange inte med parametrar) givet följande tillstånd (allt som inte är med är falskt): $\{block\ A\}; \{block\ B\}; \{goal\ (clear\ A)\};$ (on A GROUND), (on B A), (empty ARM), (clear B)

4. Hur ser tillståndet $\{block\ A\}; \{block\ B\}; \{block\ C\}; \{goal\ (clear\ A)\};$ (on A GROUND), (on B A), (on C B), (empty ARM), (clear C)} ut efter att clearBlock med parametern A har utförts (stryk över predikaten som blir falska och skriv upp de som blir samma nedan)?

Frågor efter första PDDL-filen

1. Hur enkelt var det för dig att första PDDL-filen?

| | | | | | | | | | |
|-------|--------------------------|--------------|--------------------------|--------------|--------------------------|-------------|--------------------------|------|--------------------------|
| svårt | <input type="checkbox"/> | ganska svårt | <input type="checkbox"/> | varken eller | <input type="checkbox"/> | ganska lätt | <input type="checkbox"/> | lätt | <input type="checkbox"/> |
|-------|--------------------------|--------------|--------------------------|--------------|--------------------------|-------------|--------------------------|------|--------------------------|
2. Hur svårt var det att se vad PDDL-filen modellerade?

| | | | | | | | | | |
|--------|--------------------------|---------------|--------------------------|--------------|--------------------------|--------------|--------------------------|-------|--------------------------|
| enkelt | <input type="checkbox"/> | ganska enkelt | <input type="checkbox"/> | varken eller | <input type="checkbox"/> | ganska svårt | <input type="checkbox"/> | svårt | <input type="checkbox"/> |
|--------|--------------------------|---------------|--------------------------|--------------|--------------------------|--------------|--------------------------|-------|--------------------------|
3. Hur väl strukturerad var PDDL-filen enligt dig?

| | | | | | | | | | |
|---------------|--------------------------|----------------------|--------------------------|--------------|--------------------------|---------------------|--------------------------|--------------|--------------------------|
| ostrukturerad | <input type="checkbox"/> | ganska ostrukturerad | <input type="checkbox"/> | varken eller | <input type="checkbox"/> | ganska strukturerad | <input type="checkbox"/> | strukturerad | <input type="checkbox"/> |
|---------------|--------------------------|----------------------|--------------------------|--------------|--------------------------|---------------------|--------------------------|--------------|--------------------------|
4. Hur väl var namngivningen kopplad till domänen PDDL-filen modellerar?

| | | | | | | | | | |
|--------------|--------------------------|---------------------|--------------------------|----------|--------------------------|------------------------|--------------------------|-----------------|--------------------------|
| bra kopplade | <input type="checkbox"/> | ganska bra kopplade | <input type="checkbox"/> | neutralt | <input type="checkbox"/> | ganska dåligt kopplade | <input type="checkbox"/> | dåligt kopplade | <input type="checkbox"/> |
|--------------|--------------------------|---------------------|--------------------------|----------|--------------------------|------------------------|--------------------------|-----------------|--------------------------|
5. Hur beskrivande var namngivningen för handlingarna, predikaten, variablerna och konstanterna i PDDL-filen enligt dig?

| | | | | | | | | | |
|------------|--------------------------|-------------------|--------------------------|-----------------|--------------------------|----------------------|--------------------------|---------------|--------------------------|
| visledande | <input type="checkbox"/> | ganska visledande | <input type="checkbox"/> | tillförde inget | <input type="checkbox"/> | ganska självklarande | <input type="checkbox"/> | självklarande | <input type="checkbox"/> |
|------------|--------------------------|-------------------|--------------------------|-----------------|--------------------------|----------------------|--------------------------|---------------|--------------------------|
6. Hur svarförstådd var PDDL-filen enligt dig?

| | | | | | | | | | |
|--------------|--------------------------|---------------------|--------------------------|--------------|--------------------------|---------------------|--------------------------|--------------|--------------------------|
| lättförstådd | <input type="checkbox"/> | ganska lättförstådd | <input type="checkbox"/> | varken eller | <input type="checkbox"/> | ganska svarförstådd | <input type="checkbox"/> | svårförstådd | <input type="checkbox"/> |
|--------------|--------------------------|---------------------|--------------------------|--------------|--------------------------|---------------------|--------------------------|--------------|--------------------------|
7. Hur tydligt var det vilken domän PDDL-filen modellerade?

| | | | | | | | | | |
|----------|--------------------------|-----------------|--------------------------|--------------|--------------------------|------------------|--------------------------|-----------|--------------------------|
| otydligt | <input type="checkbox"/> | ganska otydligt | <input type="checkbox"/> | varken eller | <input type="checkbox"/> | ganska uppenbart | <input type="checkbox"/> | uppenbart | <input type="checkbox"/> |
|----------|--------------------------|-----------------|--------------------------|--------------|--------------------------|------------------|--------------------------|-----------|--------------------------|
8. Hur rörig var PDDL-filen enligt dig?

| | | | | | | | | | |
|--------|--------------------------|---------------|--------------------------|--------------|--------------------------|--------------|--------------------------|-------|--------------------------|
| ordnad | <input type="checkbox"/> | ganska ordnad | <input type="checkbox"/> | varken eller | <input type="checkbox"/> | ganska rörig | <input type="checkbox"/> | rörig | <input type="checkbox"/> |
|--------|--------------------------|---------------|--------------------------|--------------|--------------------------|--------------|--------------------------|-------|--------------------------|
9. Hur lik var syntaxen för de extra handlingarna och orignalthandlingarna?

| | | | | | | | | | |
|------------|--------------------------|-------------------|--------------------------|---------------|--------------------------|----------------------|--------------------------|---------------|--------------------------|
| motstridig | <input type="checkbox"/> | ganska motstridig | <input type="checkbox"/> | varken endera | <input type="checkbox"/> | ganska överstämmande | <input type="checkbox"/> | överstämmande | <input type="checkbox"/> |
|------------|--------------------------|-------------------|--------------------------|---------------|--------------------------|----------------------|--------------------------|---------------|--------------------------|
10. Hur missvisande var namnen för handlingarna, predikaten, variablerna och konstanterna i PDDL-filen enligt dig?

| | | | | | | | | | |
|-------------|--------------------------|--------------------|--------------------------|------------------|--------------------------|--------------------|--------------------------|-------------|--------------------------|
| beskrivande | <input type="checkbox"/> | ganska beskrivande | <input type="checkbox"/> | inte missvisande | <input type="checkbox"/> | ganska missvisande | <input type="checkbox"/> | missvisande | <input type="checkbox"/> |
|-------------|--------------------------|--------------------|--------------------------|------------------|--------------------------|--------------------|--------------------------|-------------|--------------------------|

Andra PDDL-filen

På kommande sidor kommer du få se en PDDL fil och på sidan därefter finns det 4 frågor angående PDDL-filen. Svara på frågorna så exakt du kan. Det är helt fritt att bläddra tillbaka till PDDL-filen på nästkommande sida när du besvarar frågorna. Glöm inte att anteckna den aktuella tiden innan du börjar.

```

1  ;; This is a domain file modeling the blocks-world where a robot arm stacks
2  ;; blocks on the ground and on each other so that desirable towers are created.
3
4  (define (domain blocks-world)
5    (requirements :strips :adi :equality)
6
7    ;; Constants on second and third row are internal for control structures of
8    ;; the composite actions
9    (:constants CLEAR-BLOCK-WHILE STATEMENT1
10     CLEAR-BLOCK-WHILE-STATEMENT2)
11
12    ;; Predicates on first row are modeling predicates
13    ;; the composite actions
14    (:predicates (on ?b ?x) (clear ?b) (block ?b) (holding ?b) (empty ?a)
15     (clearing ?b) (rm ?and)))
16
17    ;; Internal predicate that checks if ?top is the block that is on top of
18    ;; the stack ?bottom is in
19    (define-predicate topOfStack
20      (parameters (?bottom ?top)
21        (block ?bottom) (block ?top)
22        (or (on ?top ?bottom)
23            (exists (middle)
24              (and (on ?middle ?bottom)
25                  (topOfStack ?middle ?top))
26              )))
27      )
28
29    ;; Internal predicate that checks if the planner is in a composite action
30    (define-predicate inCompositeAction
31      (parameters (?arm)
32        (clear ?b) (on ?b ?on) (empty ARM)
33        (not (inCompositeAction)))
34      )
35
36    ;; Primitive actions :::
37
38    (action pickUp
39      (parameters (?b ?on)
40        (block ?b) (or (block ?on) (= ?on GROUND))
41        (precondition (and (clear ?b) (on ?b ?on) (empty ARM)
42          (not (inCompositeAction))))
43        (effect (and (holding ?b) (clear ?on) (not (on ?b ?on))
44          (not (empty ARM))))
45      )
46
47    (action putDown
48      (parameters (?b ?on)
49        (block ?b) (or (and (block ?on) (clear ?on))
50          (= ?on GROUND))
51        (precondition (and (block ?b) (or (and (block ?on) (clear ?on))
52          (= ?on GROUND))
53          (not (empty ARM)))
54        (holding ?b) (not (empty ARM)))
55        (effect (and (not (holding ?b)) (on ?b ?on) (not (clear ?on))
56          (empty ARM)))
57      )
58
59    ;; Composite actions :::
60
61    (action clearBackstack
62      (parameters (?b)
63        (block ?b) (not (clear ?b)) (goal (clear ?b))
64        (precondition (and (not (inCompositeAction)) (empty ARM))
65        (effect (and (rm CLEAR-BLOCK-WHILE) (clearing ?b))
66      )
67
68    (action clearBlockCompositeActionBegin
69      (parameters (?b)
70        (block ?b) (not (clear ?b)) (goal (clear ?b))
71        (precondition (and (not (inCompositeAction)) (empty ARM))
72        (effect (and (rm CLEAR-BLOCK-WHILE) (clearing ?b))

```

```

73
74  ;; ?b is a block if (clearing ?b) is true since it can only be set to
75  ;; true for blocks in the previous actions
76  (action clearBlockWhileContinue
77    (parameters (?b)
78      (block ?b) (not (clear ?b))
79      (precondition (and (not (clear ?b))
80        (rm CLEAR-BLOCK-WHILE)) (rm WHILE-STATEMENT1))
81    )
82
83  ;; action also serves as the exit of the composite action
84  ;; true for blocks in the previous actions
85  (action clearBlockWhileStatement1
86    (parameters (?b)
87      (block ?b) (not (clear ?b))
88      (precondition (and (rm CLEAR-BLOCK-WHILE) (clearing ?b) (clear ?b))
89        (effect (and (not (rm CLEAR-BLOCK-WHILE)) (not (clearing ?b)))
90      )
91    )
92
93  ;; ?b is a block if (clearing ?b) is true since it can only be set to
94  ;; true for blocks in the previous actions
95  (action clearBlockWhileStatement2
96    (parameters (?b ?onTop ?secondFromTop)
97      (block ?b) (clearing ?b)
98      (block ?onTop) (topOfStack ?onTop ?b)
99      (block ?secondFromTop) (on ?onTop ?secondFromTop))
100     (effect (and (not (rm WHILE-STATEMENT1)) (rm WHILE-STATEMENT2)
101       (holding ?onTop) (clear ?secondFromTop) (not (empty ARM)))
102     (not (on ?onTop ?secondFromTop)) (not (empty ARM)))
103   )
104
105  ;; ?b is a block if (clearing ?b) is true since it can only be set to
106  ;; true for blocks in the previous actions
107  (action clearBlockWhileStatement2
108    (parameters (?onTop)
109      (block ?onTop) (holding ?onTop)
110      (precondition (and (rm WHILE-STATEMENT2) (rm CLEAR-BLOCK-WHILE)
111        (not (clearing ?onTop)) (on ?onTop GROUND) (empty ARM))
112      )
113
114  ;; clearBlock composite action end :::

```

Frågor på andra PDDL-filen

- Hur många handlingar finns det totalt definierade?

- Hur många handlingar påverkar minst ett modellerande predikat (se kommentarer för predikat i PDDL-filen för vilka predikat som är modellerande)?
4 6 7
- Vilka handlingar (en. actions) är möjliga (ange inte med parametrar) givet följande tillstånd (allt som inte är med är falskt): $\{\text{block } A\}$, $\{\text{block } B\}$, $\{\text{goal } (\text{clear } A)\}$, $(\text{on } A \text{ } \textit{GROUND})$, $(\text{on } B \text{ } A)$, $(\text{empty } \textit{ARM})$, $(\text{clear } B)$

- Hur ser tillståndet $\{\text{block } A\}$, $\{\text{block } B\}$, $\{\text{goal } (\text{clear } A)\}$, $(\text{on } A \text{ } \textit{GROUND})$, $(\text{on } B \text{ } A)$, $(\text{on } C \text{ } B)$, $(\text{empty } \textit{ARM})$, $(\text{clear } C)$ ut efter att $\textit{clearBlockEntry}$ med parametern A har utförts (stryk över predikaten som blir falska och skriv upp de som blir samma nedan)?

Frågor efter andra PDDL-filen

- Hur enkelt var det för dig att förstå PDDL-filen?
svårt ganska svårt varken eller ganska lätt lätt
- Hur svårt var det att se vad PDDL-filen modellerade?
enkelt ganska enkelt varken eller ganska svårt svårt
- Hur väl strukturerad var PDDL-filen enligt dig?
ostrukturerad ganska varken eller ganska strukturerad strukturerad
- Hur väl var namngivningen kopplad till domänen PDDL-filen modellerar?
bra kopplade ganska bra neutralt ganska dåligt dåligt kopplade kopplade
- Hur beskrivande var namngivningen för handlingarna, predikaten, variablerna och konstanterna i PDDL-filen enligt dig?
visseledande ganska tillförde inget ganska självförklarande självförklarande
- Hur svårastådd var PDDL-filen enligt dig?
lättförstådd ganska varken eller ganska svårastådd svårastådd
- Hur tydligt var det vilken domän PDDL-filen modellerade?
öryddigt ganska öryddigt varken eller ganska uppenbart uppenbart
- Hur rörig var PDDL-filen enligt dig?
ordnad ganska ordnad varken eller ganska rörig rörig
- Hur lik var syraxen för de extra handlingarna och originalhandlingarna?
motstridig ganska motstridig varken endera ganska överstämmande överstämmande
- Hur missvisande var namnen för handlingarna, predikaten, variablerna och konstanterna i PDDL-filen enligt dig?
beskrivande ganska inte missvisande ganska missvisande missvisande

Understandability of PDDL

Background Questions

Experiment outline

In the experiment you will be asked to submit some background information. Following that, you'll read a short introduction about composite actions, an extension to PDDL. In addition to that, you will be presented with a short description of the domain that is used in the experiment and a graph describing a composite action that will be used later on. Feel free to remove the page with the graph and use it as you want during the experiment. When you have read all the introductions and descriptions you'll study a PDDL-file and answer questions about it. Following that is some more questions to answer. The process will then be repeated for one more PDDL-file.

Consent form

I agree to participate in the study led by Erik Hansson at Linköping University.

I am fully aware that I am not required to do the experiment and that I can abort the session at any time if I feel any discomforts during the experiment. All the data that is gathered will be anonymous for all people outside the research team. The result of the study will be presented when the report is published. All participants have the right to get their own results.

Please sing below to state that you have read and understood the information above and that you have gotten all your questions about the session answered.

Date: _____

Please sign your name: _____

Please print your name: _____

Thank you for participating.

Contact information:

Erik Hansson

erih1172@student.liu.se

1. I am a:

Man Woman

2. Which program do you study? _____

3. Which semester are you registered in? _____

4. Have you previously encountered the subject planning within artificial intelligence?

Yes No

5. Have you used PDDL or a similar language before?

Yes No

Introduction to composite actions

Composite actions is a way to express a block consisting of actions and other blocks of actions so the planner can use them together. The way of expressing a block is similar to most programming language. For example, it could be through an if-statement, a while-loop, a sequence of actions or a single action. The syntax composite actions in PDDL is similar to the syntax of a standard actions and is as follows:

```
(:compositeAction name
  :parameters (list of variables)
  :precondition (logical expression)
  :action (action or task)
  :body (logical expression))
```

The new elements are "body", which is a block consisting of actions and blocks of actions, and "pledge" which is a logical expression that describes what the composite action promises to be true after it has been applied. One should note that the composite action can change more than what is defined in "pledge".

A block of the type *type* is expressed on the form:

```
(type
  <with (list of variables)>
  <where (logical expression)>
  type specific inputs)
```

Where "with" is a non-deterministic choice of parameters that the planner does every time the block is applied and "where" is a logic expression that states what must be true for the parameters. These two are optional to include. However, there are some type specific properties that are required. For example, the type "sequence" (a linear sequence of actions) and the type "while" both have "body" which is a list of actions blocks of actions. Moreover, the types "if" and "while" need to include "condition" which describes what is tested, "if" also needs "then" which is a block that is to be applied if "condition" is true. In addition, "if" may also have the "else" that is a block that is to be applied if "condition" is false. Finally, a special type is the "singleAction" which allows for a "with" and a "where" for a single action specified with ".do".

Complex predicates

Complex predicates are introduced in addition to complex action to abstract logical expressions and to get higher expressiveness than those in PDDL. They have the following syntax:

```
(:definePredicate name
  :parameters (list of variables)
  :predicate (logical expression))
```

Just like the standard predicates, these can be used in logical expression with the name and the parameters to the predicate. One benefit with these are that they express recursion. For example, *predicate (and (isTrue ?x) (name ?y))*. Otherwise, the work just like any predicate that is tested through *(name ?x1 ?x2 ... ?xn)*.

The goal function

It is not uncommon to guide a planner through referring to the goal in the problem definition when writing a domain file. This is useful since it is unnecessary to apply an

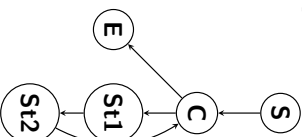
3

action if it neither provides new opportunities nor manages to fulfill something that is a goal. To specify this, a goal function is usually used. It takes one parameter that is a logical expression and tests if that exists in the goals for the problem. For example, the predicate *(goal (isTrue ?x))* is only true if *(isTrue ?x)* is one of the goals in the problem when ?x is set to the value it has.

The PDDL domain

In the experiment, both PDDL-files will model the same domain. The domain in question is the blocks-world which consists of block, a robot arm and the ground. The robot arm can be used to pick up a block that does not have anything on it and to put down the block it holds. Finally, the problems in the blocks-world consist of building one or more towers of blocks give one or more towers of blocks.

To guide the planner one composite action, *clearBlock*, has been added to the domain. The composite action clears a block by moving all the blocks on top of it. The following graph describes the control structure of *clearBlock*.



4

The first PDDL file

On the following slide you will be presented with a PDDL file and a page with 4 questions about the PDDL file. Answer the questions as exactly as you can. You are free to go back to the PDDL file when answering the questions. Do not forget to write down the current time before you start.

```

1 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
11 11
12 12
13 13
14 14
15 15
16 16
17 17
18 18
19 19
20 20
21 21
22 22
23 23
24 24
25 25
26 26
27 27
28 28
29 29
30 30
31 31
32 32
33 33
34 34
35 35
36 36
37 37
38 38
39 39
40 40
41 41
42 42
43 43
44 44
45 45
46 46
47 47
48 48
49 49
50 50
51 51
52 52
53 53
54 54
55 55
56 56
57 57
58 58
59 59
60 60
61 61
62 62
63 63
64 64
65 65
66 66
67 67
)

;; This is a domain file modeling the block-world where a robot arm stacks
;; blocks on the ground and on each other so that desirable towers are created.
(define (domain block-world)
  (:requirements :strips :equality)
  (:constants GROUND ARM)

  ;; All predicates are modeling predicates
  ;; predicates (on ?x ?y) (atom %s) (block %s) (holding ?s) (empty ?s))
  ;; Internal predicate that checks if ?top is the block that is on top of
  ;; the stack ?bottom is in
  (define-predicate topOfStack
    :parameters (?bottom ?top)
    :predicate (and (block ?bottom) (block ?top)
                    (or (on ?top ?bottom)
                        (exists (and (block ?middle)
                                     (on ?middle ?bottom)
                                     (topOfStack ?middle ?top))))))

  ;; Primitive actions
  (:action pickUp
    :parameters (?b ?an)
    :precondition (and (block ?b) (or (block ?an) (= ?an GROUND)))
    :effect (and (holding ?b) (clear ?an) (empty ARM)))
  (:action putDown
    :parameters (?b ?an)
    :precondition (and (block ?b) (or (and (block ?an) (clear ?an))
                                       (= ?an GROUND)))
    :effect (and (not (holding ?b)) (on ?b ?an) (not (clear ?an))
                 (empty ARM)))

  ;; Composite actions
  (:action clearBlock
    :parameters (?b)
    :precondition (and (block ?b) (not (clear ?b))) (empty ARM)
    :body (while
            :condition (not (clear ?b))
            :with (?aTop ?secondFromTop)
            :where (and (block ?aTop)
                       (block ?secondFromTop)
                       (topOfStack ?aTop ?b)
                       (on ?aTop ?secondFromTop)
                       (putDown ?aTop GROUND)))
          :pledge (clear ?b)
    )
  )

```

Questions about the first PDDL file

- How many actions are defined in total?

- How many actions affect at least one modeling predicate (see the comments in the PDDL-file for which predicates that are modeling)?
1 2 3
- Which actions are possible (don't supply the parameters) given the following state (everything that isn't included is false): $\{block\ A\}, \{block\ B\}, \{goal\ (clear\ A)\}$. (on *A GROUND*), (on *B A*), (empty *ARM*), (clear *B*)

- Given the state $\{block\ A\}, \{block\ B\}, \{block\ C\}, \{goal\ (clear\ A)\}$. (on *A GROUND*), (on *B A*), (on *C B*), (empty *ARM*), (clear *C*), what will the next state be after the *clearBlock* with the parameter *A* have been applied (cross out the predicates that change to false in the question text and write down those that become true)?

Post-Questions on the first PDDL file

- How easy was it for you to understand the PDDL-file?
hard quite hard neither hard nor easy quite easy easy
- How hard was it to see what the PDDL-file was modeling?
easy quite easy neither hard nor easy quite hard hard
- How good was the structure of the PDDL-file according to you?
bad quite bad neither bad nor good quite good good
- How good was the connection between the naming in the PDDL-file and what it models?
good quite good neither good nor bad quite bad bad
- How descriptive was the naming of actions, predicates, variables and constants in the PDDL-file according to you?
misguiding quite misguiding neither misguiding nor self explanatory quite self explanatory self explanatory
- How difficult was it to understand the PDDL-file according to you?
easy quite easy neither easy nor hard quite hard hard
- How clear was it which domain the PDDL-file modeled?
unclear quite unclear neither clear nor unclear quite clear clear
- How unstructured were the PDDL-file according to you?
structured quite structured neither unstructured nor structured quite unstructured unstructured
- How similar was the syntax for the extra actions to the syntax of the original actions?
conflicting quite conflicting neither consistent nor conflicting quite consistent consistent
- How misleading was the names for actions, predicates, variables and constants in the PDDL-file according to you?
descriptive quite descriptive neither descriptive nor descriptive quite descriptive descriptive

The second PDDL file

On the following slide you will be presented with a PDDL file and a page with 4 questions about the PDDL file. Answer the questions as exactly as you can. You are free to go back to the PDDL file when answering the questions. Do not forget to write down the current time before you start.

```

1 1 This is a domain file modeling the blocks-world where a robot arm stacks
2 blocks on the ground and on each other so that desirable towers are created.
3
4 (define (domain blocks-world)
5   (:requirements :strips :ndi :equality)
6
7   ;; Constants on second and third row are internal for control structures of
8   ;; the composite actions
9   (:constants CLEAR-BLOCK WHILE-CLEAR-BLOCK-WHILE-STATEMENT-1
10    CLEAR-BLOCK-WHILE-STATEMENT-2)
11
12   ;; Predicates on first row are modeling predicates
13   ;; Predicates on second row are internal for control structures of
14   ;; the composite actions
15   (:predicates (on ?b ?y) (block ?b) (holding ?b) (empty ?a)
16    (clearing ?b) (run ?and))
17
18   ;; Internal predicate that checks if ?top is the block that is on top of
19   ;; the stack ?bottom is in
20   (:definePredicate topOfStack
21     :parameters (?bottom ?top)
22     :predicate (and (block ?bottom) (block ?top)
23      (or (on ?top ?bottom)
24        (exists (?middle)
25          (and (on ?middle ?bottom)
26            (topOfStack ?middle ?top))
27          )))
28
29
30
31
32
33   ;; Internal predicate that checks if the planner is in a composite action
34   (:definePredicate inCompositeAction
35     :predicate (or (run CLEAR-BLOCK-WHILE-STATEMENT-1)
36      (run CLEAR-BLOCK-WHILE-STATEMENT-2))
37
38
39
40
41   ;; Primitive actions :::
42
43   (:action pickUp
44     :parameters (?b ?on)
45     :precondition (and (block ?b) (or (block ?on) (= ?on GROUND))
46      (clear ?b) (on ?b ?on) (empty ARM))
47     :effect (and (holding ?b) (clear ?on) (not (on ?b ?on))
48      (not (empty ARM)))
49
50
51   (:action putDown
52     :parameters (?b ?on)
53     :precondition (and (block ?b) (or (and (block ?on) (clear ?on))
54      (holding ?b) (not (empty ARM))
55      (not (inCompositeAction))))
56     :effect (and (not (holding ?b)) (on ?b ?on) (not (clear ?on))
57      (empty ARM))
58
59
60   ;; Composite actions :::
61
62   ;; clearBlock composite action begin ::
63   :clearBlock composite action begin ::
64   : Clear a block that is supposed to be clear according to the goal but :
65   : isn't at the moment. Puts all the blocks on top on the ground :
66
67   (:action clearBlockStart
68     :parameters (?b)
69     :precondition (and (block ?b) (not (clear ?b)) (goal (clear ?b))
70      (not (inCompositeAction))) (empty ARM))
71     :effect (and (run CLEAR-BLOCK-WHILE) (clearing ?b))
72

```

```

73 : ?b is a block if (clearing ?b) is true since it can only be set to
74 : true for blocks in the previous actions
75 (:action clearBlockWhileCondtrue
76 : parameters (?b)
77 : precondition (and (not (clear ?b)))
78 : effect (and (not (rm CLEAR-BLOCK-WHILE)) (rm WHILE-STATEMENT-1))
79 )
80
81 : action also serves as the exit of the composite action
82 : true for blocks in the previous actions
83 : true for blocks in the previous actions
84 (:action clearBlockWhileClear
85 : parameters (?b)
86 : precondition (and (rm CLEAR-BLOCK-WHILE)) (not (clearing ?b)))
87 : effect (and (not (rm CLEAR-BLOCK-WHILE)) (not (clearing ?b)))
88 )
89
90 : ?b is a block if (clearing ?b) is true since it can only be set to
91 : true for blocks in the previous actions
92 (:action clearBlockWhileClear2
93 : parameters (?b)
94 : precondition (and (rm WHILE-STATEMENT-1)
95 : (clearing ?b)
96 : (block ?onTop) (topOfStack ?onTop ?b)
97 : (on ?onTop ?secondFromTop))
98 : effect (and (not (rm WHILE-STATEMENT-1)) (rm WHILE-STATEMENT-2)
99 : (holding ?onTop) (clear ?secondFromTop)
100 : (not (on ?onTop ?secondFromTop))) (not (empty ARM)))
101 )
102
103 : ?b is a block if (clearing ?b) is true since it can only be set to
104 : true for blocks in the previous actions
105 (:action clearBlockWhileStatement2
106 : parameters (?onTop)
107 : precondition (and (rm WHILE-STATEMENT-2) (holding ?onTop))
108 : effect (and (not (rm WHILE-STATEMENT-2)) (rm CLEAR-BLOCK-WHILE)
109 : (not (holding ?onTop)) (on ?onTop GROUND)) (empty ARM))
110 )
111
112 ::: clearBlock composite action end :::
113
114

```

Questions about the second

1. How many actions are defined in total?

| | | | | | |
|---|--------------------------|---|--------------------------|---|--------------------------|
| 4 | <input type="checkbox"/> | 6 | <input type="checkbox"/> | 7 | <input type="checkbox"/> |
|---|--------------------------|---|--------------------------|---|--------------------------|
2. How many actions affect at least one modeling predicate (see the comments in the PDDL-file for which predicates that are modeling)?

| | | | | | |
|---|--------------------------|---|--------------------------|---|--------------------------|
| 4 | <input type="checkbox"/> | 6 | <input type="checkbox"/> | 7 | <input type="checkbox"/> |
|---|--------------------------|---|--------------------------|---|--------------------------|
3. Which actions are possible (don't supply the parameters) given the following state (everything that isn't included is false): $\{block\ A\}, \{block\ B\}, \{goal\ (clear\ A)\}, (on\ A\ GROUND), (on\ B\ A), (empty\ ARM), (clear\ B)\}$

4. Given the state $\{block\ A\}, \{block\ B\}, \{block\ C\}, \{goal\ (clear\ A)\}, (on\ A\ GROUND), (on\ B\ A), (on\ C\ B), (empty\ ARM), (clear\ C)\}$, what will the next state be after the *clearBlockEntry* with the parameter *A* have been applied (cross out the predicates that change to false in the question text and write down those that become true)?

Post-Questions on the second PDDL file

1. How easy was it for you to understand the PDDL-file?

| | | | | |
|-------------------------------|-------------------------------------|--|-------------------------------------|-------------------------------|
| <input type="checkbox"/> hard | <input type="checkbox"/> quite hard | <input type="checkbox"/> neither hard nor easy | <input type="checkbox"/> quite easy | <input type="checkbox"/> easy |
|-------------------------------|-------------------------------------|--|-------------------------------------|-------------------------------|

2. How hard was it to see what the PDDL-file was modeling?

| | | | | |
|-------------------------------|-------------------------------------|--|-------------------------------------|-------------------------------|
| <input type="checkbox"/> easy | <input type="checkbox"/> quite easy | <input type="checkbox"/> neither hard nor easy | <input type="checkbox"/> quite hard | <input type="checkbox"/> hard |
|-------------------------------|-------------------------------------|--|-------------------------------------|-------------------------------|

3. How good was the structure of the PDDL-file according to you?

| | | | | |
|------------------------------|------------------------------------|---|-------------------------------------|-------------------------------|
| <input type="checkbox"/> bad | <input type="checkbox"/> quite bad | <input type="checkbox"/> neither bad nor good | <input type="checkbox"/> quite good | <input type="checkbox"/> good |
|------------------------------|------------------------------------|---|-------------------------------------|-------------------------------|

4. How good was the connection between the naming in the PDDL-file and what it models?

| | | | | |
|-------------------------------|-------------------------------------|---|------------------------------------|------------------------------|
| <input type="checkbox"/> good | <input type="checkbox"/> quite good | <input type="checkbox"/> neither good nor bad | <input type="checkbox"/> quite bad | <input type="checkbox"/> bad |
|-------------------------------|-------------------------------------|---|------------------------------------|------------------------------|

5. How descriptive was the naming of actions, predicates, variables and constants in the PDDL-file according to you?

| | | | | |
|-------------------------------------|---|--|---|---|
| <input type="checkbox"/> misguiding | <input type="checkbox"/> quite misguiding | <input type="checkbox"/> neither misguiding nor self explanatory | <input type="checkbox"/> quite self explanatory | <input type="checkbox"/> self explanatory |
|-------------------------------------|---|--|---|---|

6. How difficult was it to understand the PDDL-file according to you?

| | | | | |
|-------------------------------|-------------------------------------|--|-------------------------------------|-------------------------------|
| <input type="checkbox"/> easy | <input type="checkbox"/> quite easy | <input type="checkbox"/> neither easy nor hard | <input type="checkbox"/> quite hard | <input type="checkbox"/> hard |
|-------------------------------|-------------------------------------|--|-------------------------------------|-------------------------------|

7. How clear was it which domain the PDDL-file modeled?

| | | | | |
|----------------------------------|--|--|--------------------------------------|--------------------------------|
| <input type="checkbox"/> unclear | <input type="checkbox"/> quite unclear | <input type="checkbox"/> neither clear nor unclear | <input type="checkbox"/> quite clear | <input type="checkbox"/> clear |
|----------------------------------|--|--|--------------------------------------|--------------------------------|

8. How unstructured were the PDDL-file according to you?

| | | | | |
|-------------------------------------|---|--|---|---------------------------------------|
| <input type="checkbox"/> structured | <input type="checkbox"/> quite structured | <input type="checkbox"/> neither unstructured nor structured | <input type="checkbox"/> quite unstructured | <input type="checkbox"/> unstructured |
|-------------------------------------|---|--|---|---------------------------------------|

9. How similar was the syntax for the extra actions to the syntax of the original actions?

| | | | | |
|--------------------------------------|--|---|---|-------------------------------------|
| <input type="checkbox"/> conflicting | <input type="checkbox"/> quite conflicting | <input type="checkbox"/> neither consistent nor conflicting | <input type="checkbox"/> quite consistent | <input type="checkbox"/> consistent |
|--------------------------------------|--|---|---|-------------------------------------|

10. How misleading was the names for actions, predicates, variables and constants in the PDDL-file according to you?

| | | | | |
|--------------------------------------|--|--|--|------------------------------------|
| <input type="checkbox"/> descriptive | <input type="checkbox"/> quite descriptive | <input type="checkbox"/> neither descriptive nor descriptive | <input type="checkbox"/> quite deceptive | <input type="checkbox"/> deceptive |
|--------------------------------------|--|--|--|------------------------------------|



PDDL Domain Introduction

Swedish version used in the pre-experiment

PDDL kort intro

En beskrivning, i PDDL, till en planerare består vanligen av två filer: en domänfil och problemfil. Domänfilen som beskriver vad som kan finnas, hur olika objekt kan förhålla sig till varandra, vilka konstanter som finns i domänen osv:et vilket problem som ska lösas samt vilka handlingar som kan göras i domänen. Problemfilen som beskriver vilka objekt som finns, hur starttillståndet ser ut och vad måltillståndet är. Det här dokumentet ger en kort översikt över hur en domänfil i PDDL är uppbyggd.

Domänfilens syntax

En domänfil innehåller vanligen fem olika komponenter: namnet på domänen, de moduler som används i domänfilen, de konstanta sakerna i domänen, vilka förhållanden som gäller (predikat) samt vilka handlingar som finns. I en PDDL domänfil skrivs det som följande:

```
(define (domain namn)
  (:requirements ) ;; moduler
  (:constants ) ;; konstanter
  (:relations between objects
  (:action) ;; one action in the domain
)
```

Namnet är en sträng som beskriver vad domänen heter. requirements, constants och predicates är listor över moduler, strängar respektive predikat. En action är antingen mer komplicerad men vi återkommer till det lite senare. Först är det vart att säga vad ett predikat är. I PDDL är ett predikat namngivet, har ett godtyckligt antal parametrar och är sant eller falskt. Ett exempel är predikatet

```
(uw ?x)
```

Som modellerar att ett objekt är en uw (sant) eller inte är en uw (falskt).

Handlingars syntax

En action är en handling som kan användas av planeraren för att ändra det nuvarande tillståndet. De specificeras på följande sätt (illustrerat med ett exempel):

```
(:action flyTo
  :parameters (?u ?from ?to)
  :precondition (and (uw ?u) (location ?from) (location ?to)
                    (not (= ?from ?to)) (not (sak ?u ?to))
                    (sak ?u ?from)) (sak ?u ?to))
  :effect (and (not (sak ?u ?from)) (sak ?u ?to))
)
```

Det här exemplet är en handling som förflyttar en uw från en plats till en annan. :parameters beskriver vilka parametrar som handlingen tar (i det här fallet tre parametrar); :precondition beskriver vad som måste vara sant i det nuvarande tillståndet och :effect är det som kommer vara sant i tillståndet efter att handlingen har utförts (de som inte benämnas kommer inte att förändras). Både :precondition och :effect beskrivs med logiska uttryck. Formen på ett logiskt uttryck är antingen ett predikat eller en funktion som beskrivs med en parentes där första tokenen är funktionen som antogs och resterande token i parentesen är parametrar (logiska uttryck).

I PDDL kan man använda sig av många olika logiska funktioner för att specificera ett logiskt uttryck. Bland de vanligaste är:

- and: sann om alla parametrarna (godtyckligt antal) är samma.
- or: sann om en av parametrarna (godtyckligt antal) är samma.
- exists: sann om det existerar variabler (en lista specificerad som första parametern) så att ett logiskt uttryck blir sant (andra parametern).
- forall: sann om ett logiskt uttryck (andra parametern) blir sant för alla möjliga instanser av en mängd variabler (en lista specificerad som första parametern).
- =: sann om första och andra parametern är lika.
- not: sann om parametern är falsk.

PDDL short introduction

A description, in PDDL, to a planner usually consists of two files, one domain file and one problem file. The domain file describes what can exist, how the different objects relate to each other, which constants exist in the domain for all problems and which actions that can be applied in the domain. The problem file describes which objects do exist, what the start state is and what the goal state is. This document gives a short description of a domain file in PDDL.

The syntax of a domain file

A domain file usually consists of five different components: the name of the domain, the modules that are used in the domain file, the constant that is in the domain, which relations exist (predicates) and which actions exist. In PDDL, this is expressed in the following way:

```
(define (domain name)
  (:requirements ) :: modules
  (:constants ) :: relations between objects
  (:predicates ) :: relations between objects
  (:action ) :: one action in the domain
)
```

The name is a string that describes the name of the domain, requirements, constants and predicates are lists of modules, strings respectively predicates. An action is a bit more complex but we will come back to that later. First we will describe what a predicate is. In PDDL, a predicate is named, has an arbitrary amount of parameters and is either true or false. For example, the following is a predicate that models that an object is a UAV (true) or not a UAV (false):

```
(uav ?u)
```

The syntax of an action

An action is something that the planner can apply to change the current state. They are specified in the following way (illustrated with an example):

```
(:action flyTo
  :parameters (?u ?from ?to)
  :precondition (and (not (= ?from ?to)) (location ?from) (location ?to)
                    (isAt ?u ?from)) (not (isAt ?u ?to))
  :effect (and (not (isAt ?u ?from)) (isAt ?u ?to)))
)
```

The example is an action that moves a UAV from one location to another. :parameters describes the parameters of the action (in this case three parameters). :precondition describes what must be true in the current state and :effect is what is true in the state after the action has been applied (everything that is not mentioned will remain as it was previously). Both :precondition and :effect is described with a logical expression. A logical expression is either a predicate or a function that is written with a parenthesis in which the first token is the name of the function that is called and the rest of the tokens are the arguments (logical expressions).

It is possible to use quite a lot of different logical functions to specify a logical expression in PDDL. Among the most common are:

- and: true if all the parameters (an arbitrary amount) are true.
- or: true if one of the parameters (an arbitrary amount) are true.
- exists: true if there exist variables (a list specified by the first parameter) such that a logical expression (second parameter) is true.
- forall: true if a logical expression (second parameter) is true for all possible instances of a list of variables (a list specified by the first parameter).
- =: true if the first and the second parameter are equal.
- not: true if the parameter is false.



E Search Time - Compared to no Guidance

| Execution number | Run time on CPU (ms) |
|------------------|----------------------|
| 1 | 2 |
| 2 | 2 |
| 3 | 2 |
| 4 | 1 |
| 5 | 2 |
| 6 | 2 |
| 7 | 1 |
| 8 | 2 |
| 9 | 1 |
| 10 | 1 |
| 11 | 2 |
| 12 | 1 |
| 13 | 1 |
| 14 | 1 |
| 15 | 1 |
| 16 | 1 |
| 17 | 1 |
| 18 | 1 |
| 19 | 2 |
| 20 | 1 |

Table E.1: Run time for domain with composite action

| Execution number | Run time on CPU (ms) |
|-------------------------|-----------------------------|
| 1 | 2643 |
| 2 | 2708 |
| 3 | 2718 |
| 4 | 2578 |
| 5 | 2625 |
| 6 | 2610 |
| 7 | 2621 |
| 8 | 2740 |
| 9 | 2708 |
| 10 | 2532 |
| 11 | 2667 |
| 12 | 2734 |
| 13 | 2815 |
| 14 | 2672 |
| 15 | 2650 |
| 16 | 2684 |
| 17 | 2522 |
| 18 | 2533 |
| 19 | 2660 |
| 20 | 2626 |

Table E.2: Run time for domain without any search guidance



F Data understandability pre-experiment

| Participant | Estimated difference | Measured difference | First domain guidance |
|-------------|----------------------|---------------------|----------------------------|
| 1 | 0.5 | 0.125 | Composite action |
| 2 | 0.5 | 0.500 | Converted composite action |
| 3 | 2.5 | 1.750 | Composite actions |
| 4 | 2.0 | 0.500 | Converted composite action |
| 5 | 0.0 | 0.375 | Composite action |

Table F.1: The data from the understandability pre-experiment. The difference are calculated by subtracting the measured value for the PDDL file that does not have composite actions from the PDDL file that has composite actions. The fourth row describes which guidance method the first PDDL file in their questionnaire had.



Data understandability experiment

| Participant | Estimated difference | Measured difference | First domain guidance |
|-------------|----------------------|---------------------|----------------------------|
| 1 | 1.5 | 1.125 | Converted composite action |
| 2 | 2.0 | 1.250 | Composite action |
| 3 | .5 | .375 | Converted composite action |
| 4 | 1.5 | .500 | Converted composite action |
| 5 | 0.0 | 1.000 | Composite action |
| 6 | 1.5 | .750 | Composite action |

Table G.1: The data from the understandability experiment. The difference are calculated by subtracting the measured value for the PDDL file that does not have composite actions from the PDDL file that has composite actions. The fourth row describes which guidance method the first PDDL file in their questionnaire had.



Bibliography

- [1] J. A. Baier, C. Fritz, and S. A. McIlraith, “Exploiting procedural domain control knowledge in state-of-the-art planners”, in *ICAPS*, 2007, pp. 26–33.
- [2] J. Baier and J. Pinto, “Integrating true concurrency into the robot programming language golog”, in *Computer Science Society, 1999. Proceedings. SCCC’99. XIX International Conference of the Chilean*, IEEE, 1999, pp. 179–186.
- [3] J. Bansiya and C. G. Davis, “A hierarchical model for object-oriented design quality assessment”, *Software Engineering, IEEE Transactions on*, vol. 28, no. 1, pp. 4–17, 2002.
- [4] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality”, in *Proceedings of the 2nd international conference on Software engineering*, IEEE Computer Society Press, 1976, pp. 592–605.
- [5] J. Brooke, “Sus-a quick and dirty usability scale”, in *Usability evaluation in industry*, London: Taylor & Francis, 1996, pp. 189–194.
- [6] J. K. Chhabra, K. Aggarwal, and Y. Singh, “Code and data spatial complexity: two important software understandability measures”, *Information and software Technology*, vol. 45, no. 8, pp. 539–546, 2003.
- [7] G. De Giacomo, Y. Lespérance, and H. J. Levesque, “Congolog, a concurrent programming language based on the situation calculus”, *Artificial Intelligence*, vol. 121, no. 1, pp. 109–169, 2000.
- [8] P. Doherty, J. Kvarnström, and A. Szalas, “Temporal composite actions with constraints”, in *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2012.
- [9] M. Genero, G. Poels, and M. Piattini, “Defining and validating metrics for assessing the understandability of entity–relationship diagrams”, *Data & Knowledge Engineering*, vol. 64, no. 3, pp. 534–557, 2008.
- [10] R. Harrison, S. Counsell, and R. Nithi, “Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems”, *Journal of Systems and Software*, vol. 52, no. 2–3, pp. 173–179, 2000.
- [11] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, “Preliminary guidelines for empirical research in software engineering”, *Software Engineering, IEEE Transactions on*, vol. 28, no. 8, pp. 721–734, 2002.

-
- [12] J. A. Krosnick and S. Presser, "Question and questionnaire design", in *Handbook of survey research*, 2nd ed., Emerald Group Publishing Limited Bingley, UK, 2010, ch. 9, pp. 263–314.
- [13] D. S. Kushwaha and A. K. Misra, "Improved cognitive information complexity measure: a metric that establishes program comprehension effort", *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 5, pp. 1–7, 2006.
- [14] J. Kvarnström, "Planning for loosely coupled agents using partial order forward-chaining", in *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 2011.
- [15] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl, "Golog: a logic programming language for dynamic domains", *The Journal of Logic Programming*, vol. 31, no. 1, pp. 59–83, 1997.
- [16] D. S. Nau, "Current trends in automated planning", *AI magazine*, vol. 28, no. 4, p. 43, 2007.
- [17] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Englewood Cliffs: Prentice Hall, 2010.
- [18] M. Shah, L. Chrupa, F. Jimoh, D. Kitchin, T. McCluskey, S. Parkinson, and M. Vallati, "Knowledge engineering tools in planning: state-of-the-art and future challenges", in *Proceedings of the 4th Workshop on Knowledge Engineering for Planning and Scheduling*, 2013, pp. 53–60.
- [19] V. Strobel and A. Kirsch, "Planning in the wild: modeling tools for pddl", in *Proceedings of the 37th German Conference on Artificial Intelligence (KI 2014)*, Springer, 2014, pp. 273–284.
- [20] Y. Wang and J. Shao, "Measurement of the cognitive functional complexity of software", in *Proceedings of the Second IEEE International Conference on Cognitive Informatics*, Aug. 2003, pp. 67–74.
- [21] D. S. Weld, "An introduction to least commitment planning", *AI magazine*, vol. 15, no. 4, p. 27, 1994.