Project .....

Artificial Intelligence in the Computer Game Astro Kid

Kasper Reindahl Rasmussen (s103476)

November 23, 2015

Contents

1 Introduction

what is planning plannig is basically having a problem and finding a series of actions that solves the problem

classical planning

what is a planner

two ways to automate the process either building a planner to the particular problem or to use a general purpose planner. the main difference .....

to automate this process the problem needs to be described in away a general planner can understand.

what is a planning language

the most common planning language for classical planning is Strips and PDDL, based around states

is a way of describing initial state, goal state and the possible actions available to manipulate the current state.

in classic planning the concept of time is missing, actions have no duration and instance effect

in reality things/actions dont always behave like this, they happen over a time span.... effect under and after, preconditions holds....

the goal is to be able to solve a game without prior or only little initial knowledge this is to be achieved in two parts first is to learn to interact with the environment and the consequences of the actions second use a general planner to solve the problem using the found knowledge

keep knowledge as it progress through levels abstract away from objects and use types

the goal isnt to learn the complete action schema just one thats good enough to solve the level

2 related works

classic planning normally operates whit action having no duration, and transition between states happening instantaneous.

and this works for many types of problems such as blocks world and sokoban

in reality things .... takes time....

the concept of actions having duration isnt anything new,

pddl durative actions....

the concept of temporal planning was introduced in PDDL 2.1, one of the first planners handling this in pddl [Garrido et al., 2002]

it has the notion of time, but its more directed towards scheduling

preconditions start under end

duration

effect start end

should end in a safe state

concurency in planning isnt either something new

various adaptation of has been suggest to integrete in a planning system and in particular MaPL wich builds on [Brenner, 2003]

3 PDDL Background

Planning Domain Definition Language is a planning language, created in 1998 by Drew McDermott and the AIPS-98 Planning Competition Committee. It was created to make the international planning competition possible and to breach gab between research and application. The development of PDDL is driven around each IPC, and the development of planners is often also effected by this. PDDL is an action based language, based on strips with some advanced extensions such as ADL. It provides a deterministic single agent discrete fully observable planning environment, it also works under the closed world assumption. The current version of PDDL is 3.1.

PDDL is separated into two parts problem and domain description. The problem description, is the description of a problem instance, which consist of the state of the world and Goal condition. The domain description is on the other hand the rule set of the world, consisting of the actions which is allowed to be performed on the world, and what type of objects is allowed to exist in the world. an example of this can be seen in Appendix A using the classic blocks domain. This separation means that a single domain description can be paired to many problems, but not necessarily the other way around.

4 The Astro kid world

The Astro Kid game is a discrete world where everything moves in a 2D grid, and the world is fully observable. Its a single agent system based around the action which can be start by controlling the avatar. These "player" actions is movement (up, down, left, right), push, remote control. These are the action which the player controls, and everything else is consequences of theses actions. That everything else is consequences means that these actions are the only ones where the planner have a choice between actions that effect the final result.

Actions that do fall into the category of consequences, can be such as a robot walking when pushed/activate, objects falling when push over an edge etc. The difference between the two groups is that the planner should not be able to stop an object from falling mid air, but it should be able to chose if the avatar moves left or right.

When looking at the Astro Kid domain, its worth noting the main areas where it differentiates from classical planner domains like the gripper. These areas are concurrent and continuous actions. Continuous actions are defined as actions that happens over several time steps. The actions with continuous actions all relates to the movement of objects (falling and sliding/walking), an example of this is that falling from the top of the map to the bottom doesn't happen instantaneous, but in steps and it has to pass through the areas in between. As a result of this multiple objects can move at the same time, which leads to concurrent actions.

The number of concurrent actions is limited, since effects/consequences of continuous actions cant increase the number of moving objects, this means that are a maximum one new action with continuous effect, that can be started each step. An example of this can be seen level 25. This level also illustrates that even though one action can be started each turn, the total number is limited by the finite size of the map. In this case a maximum of eleven stones and the

avatars can be moving at the same time, (the number of actions are increased when adding remote controls but is still limited)

5 General planning

The idea of using a general planner is that its possible to solve the problem efficiently, without building a planner from scratch for each problem. The general planer will however nearly often be less efficient than an planner for the specific problem. This is due to it not always being able recognize and utilize features that are specific to a given domain. The main question is therefore if the general planner can solve the problem efficiently enough to be useful.

To begin solving the problem with a general planner, the domain needs to be described in a Planning languages. In this case PDDL is suitable choice since it is a expressive language and used for the IPC, and there therefore exists a series of planners that support this language.

The general planner there will be used is Fast Downward. Fast Downward was chosen due to it being a widely know open source planner and it has done well in several IPCs, eg. won the Sequential Satisficing and Optimization parts of IPC 2011[1]. It must therefore be considered one of the top planers out there, even though it only support a subset (PDDL 2.2 level 1 + action cost) of the complete PDDL.

5.1 PDDL

The chosen planner is Fast Downward, which is limited to PDDL 2.2 level 1 and the action cost feature, therefore this subset of PDDL will be used.

Due to how the domain differentiate from the type of problems usually found in classical planning. There are used two different approaches to describing the domain in PDDL. The domain can either be transform/relaxed into a more classical planning domain or use the structure as is and work around the problems that occur due to it.

5.1.1 Problem representation

There are several approach to the problem, but the basic structure remains the same. The differences between the approaches lies in the actions available and the ordering of these. This is possible due to PDDLs separation Problem description and Domain description.

The main part of the problem description is the location of the different objects in the domain. This is handled by each point in the world's 2D grid is represented in PDDL as an object and its location is defined relative to its neighbours. All other objects locations in the domain are defined relative to these points.

Even though the domain description changes, how the metric is used in remains the same for both domains. The planner can either optimise toward plan length or cost. This is useful since that the found plan consist of all actions, which will inevitably contain actions, that is consequence of the players actions, the plan length will therefore differentiate from the number of player actions.

[1] `http://www.plg.inf.uc3m.es/ipc2011-deterministic/Results.html`

To filter out these actions, a cost is add to the players actions and everything else is cost zero. and thereby optimising by cost instead of length.

5.1.2 Classic Approach

What will further on be described as the "Classic Approach" is build around getting the Domain to be more similar to that of a classical planning problem. This approach centred around avoiding an input update cycle, and keeping the separation of course and effect as short as possible. The main advantage of doing this, is that general planners are optimized for this kind of problems, since that is what they usually handle at the IPC.

The main problem area with the classical approach is that PDDL doesn't directly allow concurrent actions, unless they are changed/merged into a single action. This could be done by having a action for each possible combination of objects moving. This approach would however leads to an explosion of possible actions, since the number of actions would then be depend on the number of objects that can have a continuous effect. Another problem with having the number of actions dependent on the number of objects, is that domain would then either have to be limited to a certain number objects or change depending on the number of objects in the problem description. This would be against how PDDL is intended to be used.

To avoid this problem, the domain can be relaxed by making some assumptions, so it would fit better into a classic planning paradigm. The assumption is that moving objects don't interact with each other. This means that concurrency can be avoid, by letting one of the concurrent actions completely terminate, before the next is considered. To avoid the avatar from standing still unnecessary when executing the plan, only agent actions is considered as part of the solution (when executing the plan the agent dosnt wait for concurrent actions to stop).

in the ideal world with this assumption i would make it possible to apply a single of the actions with continuous effect and treat it as a normal action due to the only interesting state being the last of the continuous action. however even with this assumption it isn't feasible to calculate which state a given continuous actions ends, due to it depending on all the steps in between.

This assumption holds in most cases due to objects in general moves away from the avatar (actions shuch as fall, slide, walk, push). Which also means that any subsequent action wont interfere with others, due to objects moving at same speed. Teleporters, remote controls and gates can in a few cases interfere with this. If this assumption holds, it would be possible to find the optimal solution.

Applying this however means that minor discrepancies between the real world and the PDDL world can occur. The differences can in some cases be ignored since, what is important, is that the avatars route is clear. If the discrepancies is to great to be ignored eg. if problems occurs between multiple moving objects (slide + robot activate, eg. level 31). The assumption can in most cases be ensured by inserting noOps into the plan after a solution is found and thereby ensuring only one thing moves at the time as the assumption requires. This however results in a not necessarily the optimal solution, and

there will still be a few instances where two moving objects needs to interact to solve the problem, if this is the case the problem cant be solved, due to the limitations of the assumption.

Using this approach means that each action needs a rather large set of precondition, this is to ensure that not only is the precondition for the given action fulfilled, but also that no other action is running/executing. The main precondition for most of the action are therefore a condition that ensures that nothing else is moving (continuous effect). In general several different things is handled in each action, and they are in most cases the same for several other actions.

There is a few cases where this isnt the case, since they are not restricted by other actions. In the case of gates opening and closing, they are handled by a derived predicate (axioms) instead. Further more to simplify the precondition and effects of the actions, part of the functionality are extracted into their own optional actions. This the case for picking up items and teleportation. They are in fact not an optional action, but due to their locations works well as optionals. And in the case of teleportation its possible to treat it as optional action, using post processing since its reversible.

5.1.3 Update Approach

Due to the problem with a few unsolvable levels in the classic approach, and the wish of obtaining good solutions without post processing. A different approach is needed. The main idea with this approach, is the idea of using the domain as it is, this is to avoid enforcing/making assumption of the domain, which then can effect the quality of the found solutions. Basically to ensure that the PDDL version of the world is at all times the same as the real world. This approach guaranties an optimal and valid solution.

When looking at the Domain there are basically two types of actions, the ones where the planner has a choice (player action) and actions which are consequences/updates of earlier actions. The main idea is therefore to separate the domain into input/choice and update, and then enforcing a ordering of the actions so a input/choice is always followed by an update. The ordering of actions can be enforced by using flags. This ordering makes it possible for the action to be limited in purpose and therefore to have fewer and simpler preconditions, and thereby making it more readable.

The update step is itself split up into multiple parts with a strict ordering, to enforce that the concurrent actions interact correctly with each others. The update step is separated into: removing objects, changing direction of objects, moving objects, collecting items, opening/closing gates, teleportation. To ensure all the continuous effects are applied on the relevant objects universal quantifiers are used.

A problem with this approach is that how the strict ordering of actions (input update cycle), differentiates from the usually approach used in general planning, this means the direct link between action and effect is not as clear as in classic PDDL. this is due to the update sequence/delaying effects, where effects of an action later applied in update phase.

6 Tests

There are various ways of tweaking the performance of the planner. The two most obvious ways of doing this, is changing the parameters of the planner and changing the PDDL domain to fit the planners strengths better.

6.1 Plan Quality

Table 1: Plan Quality

	level 4	level 7	level 25small	problem 24	problem 24v2
classic	35	45	x	11	11
update	35	45	32	7	11

Length of the found plans

Looking at results in table 1 it can be seen that on problems where the assumption holds the found solutions are equal. Some difference can be seen where the post processing is necessary (problem 24 and 24v2 where the robot is activated to late). The post processing can in most cases be optimised to perform close or equal to the update approach. However to achieve the best possible result from the post processing domain specific knowledge would be need. A greater difference is seen on levels such as level 25, where the assumptions used in the classic approach fails completely (moving objects needs to hit each other). The number of levels where this is the case is fairly limited. As one would expect it can be seen that domain only effect the quality of the solution where the assumptions made dosnt hold.

6.2 Speed

When looking at the data its worth taking note of that most of the time is not always spend in search, as one would expect for a planner, but instead in preprocessing/translating. This is interesting due to that most other planners, works more directly with the PDDL description (dont translate to SAS+), and therefore dosnt have to use the same amount of time on this step.

A significant speed advantage to the classic approach can be seen on all problems it can solve. The difference in speed was expected, due to the domain type being relaxed and more similar to the ones used in the IPC. One would therefore expect the planner to be more optimized for this kind of problem.

What wasn't expected was how large the difference. On a problem such as level 4 is the difference roughly a factor of 20, it is however worth taking note of that the speed difference isnt equal for all the sub parts of the search. It is in particular the translator which is the bottle neck.

The great speed difference makes the update approach less useful. For the it to be able to solve more levels this speed difference needs to be reduced. Test have shown that use of universal quantifiers have a large impact on the translating/instantiation time. Therefore there is created a variation of the domain where a single universal quantifier is replaced by an existential quantifier and an extra action(s).(code can be found at Appendix C)

When running the different versions of the domain on various problems, one thing becomes clear, the total times varies greatly depending on the domain

Table 2: problem 4

	forall	simple
total Time	26.2	136.2
translator Time	21.9	6.6
relevant atoms	8872	12968
auxiliary atoms	16959	18807
final queue length	25831	31775
total queue pushes	58904	68537
axioms	1	132652
peak memory	100568 KB	208356 KB
task size	42751	573269
preprocessing Time	4.2	127.4
necessary operators	7292	7337
search Time	0.1	2.2

and level combination (table 7). However when looking at the results (table 11), what shows is that the use of universal quantifiers greatly increases the instantiation/translation time.

Replacing the universal quantifiers ensures a quicker instantiation/translation, but it has the cost of the number of axioms exploding, and in general the number of atoms growing by 5-20%, which in the end can greatly hampers the preprocessing that generates the Causal graph.

The tendency seams to be that the more movable objects and complex the problem the better the version with universal quantifiers does, this is especially clear when moving from toy problem to actual levels (tabel 4 and 5). Generally this existential variation of the domain only really shines when the problem is "small" and isnt therefore useful on the different levels in Astro Kid.

Table 3: problem 4v2

	Universal	Existential
total Time	9.8	5.2
translator Time	7.1	2.2
relevant atoms	6742	8199
auxiliary atoms	13900	14710
final queue length	20642	22909
total queue pushes	40653	44321
axioms	1	534
peak memory	71356 KB	66040 KB
task size	29496	31067
preprocessing Time	2.6	2.8
necessary operators	5469	5485
search Time	0.1	0.2

An anomaly is shown in prob4v2 table 3 (where a single stone is add to problem04) and not much difference would be expected, but here the planner figures out that the objects is in fact not movable, and the explosion of axioms dosnt happen.

Another way of tweaking the code is to optimize the Problem Definition,

this can be done by removing unreachable states/objects, more precise remove the representation of position that isn't useful. The effect of this can clearly be seen when adding unused location to prob02 and the results are shown in table 6. The results shows for each variation the time need roughly doubles. Interestingly enough the version of the domain with existential quantifiers isnt effected nearly as much by it (table 7). one version scales with the size of the problem (grounding), the other with the number of objects (axiom explosion). This shows that long running times does not necessary correlate to a more complex problem. It also especially for universal quantifier version shows the weakness of a general planner, which is that it cant use specific knowledge of the domain, and therefore cant easily discard non relevant areas of the problem.

Table 4: level 4

	Universal	existential
total Time	175.4	x
translator Time	164.4	x
relevant atoms	27782	x
auxiliary atoms	40327	x
final queue length	68109	x
total queue pushes	201441	x
axioms	13	x
peak memory	231780 KB	x KB
task size	138081	x
preprocessing Time	10.0	x
necessary operators	24525	x
search Time	1.0	x

Table 5: level 9

	classic	Universal	existential
total Time	11.5	301.7	x
translator Time	5.6	280.3	x
relevant atoms	8797	61431	x
auxiliary atoms	169254	64980	x
final queue length	178051	126411	x
total queue pushes	746595	359476	x
axioms	265	23	x
peak memory	195568 KB	386872 KB	x
task size	43197	265076	x
preprocessing Time	5.4	17.2	x
necessary operators	4283	49496	x
search Time	0.5	4.2	x

6.3 Heuritics

The main parameter in Fast Downward that can be tweaked is the heuristic. This choice of heuristic is fairly limited due most of the them supported by fast downward dosnt support the use of axioms or conditional effects, which are used

Table 6: white space. for each version the width of the problem have been increased by 5

	prob02	prob02v2	prob02v3	prob02v4
total Time	11.2	24.0	49.3	91.0
translator Time	7.6	17.8	38.8	75.5
relevant atoms	12394	22257	34922	50387
auxiliary atoms	14389	20021	25656	31291
final queue length	26783	42278	60578	81678
total queue pushes	58980	101633	155153	219523
axioms	1	1	1	1
peak memory	100800 KB	157492 KB	229452	317760
task size	56625	104505	166385	242265
preprocessing Time	2.4	5.8	9.9	24.5
necessary operators	10774	20079	32184	47089
search Time	0.2	0.4	0.6	1.0

Table 7: Running times

	prob00	prob01	prob02	prob02v2	prob02v3	prob02v4	prob02v5	prob03
universal	0.892	0.887	11.326	24.798	52.617	95.08	x	0.794
existential	0.646	0.788	6.777	12.148	18.863	27.405	36.86	0.704

	prob04	prob04v2	prob07	prob09	prob10	prob11	prob12	level 4
universal	23.905	9.197	36.326	5.842	13.45	2.4 2	21.484	429.9
existential	124.794	4.434	14.228	2.444	5.789	1.713 s	7.346 s	x

in the domain description, and those heuristics that does, do so barely[2]. The only supported heuristic that is admissible is blind, and as the name suggest isnt the most advanced of the heuristics.

The importance of the choice heuristic varies greatly depending on the level. and if speed, quality or a combination is wished for. For the update approach choice of heuristic is fairly unimportant since the bottleneck is at the translator/preprocessing. When looking at the classic approach this changes. The effect of the heuristic on the different levels varies greatly, fx. on level 4 the time consumed by the search is minor.

The heuristics blind and context enhanced additive both does well on all the level except a single one each.

The additive heuristics is consistently fast but gives a solution of poor quality.

The blind heuristic gives a high quality due to it being admissible and it is fast except on a single level where it is 60 time slower than the second to last.

The simplest possible explanation for why the some heuristics does terrible on some levels, is that they cant handle axioms well and will therefore handle certain situations "stupid"[3]. This cant explain why Blind fail, due to i fully supporting all features of the domain. The reason for the failure instead lies in that its an uninformed search, which means that the no knowledge of the domain is used and is therefore prone to explore the state space less efficient. Therefore

[2](in the sense that the planner won't complain – handling of axioms might be very stupid and even render the heuristic unsafe) `http://www.fast-downward.org/Doc/Heuristic`

[3]Fast downwards description of the situation..

it is not unexpected that blind fails on a level, but more that it dosnt fails on other levels. That this does not happen, must be credited to the preprocessing of the problem.

Table 8: level 4

translate+preprocess 7.425 s		
name	time (s)	lenght
Additive	0.329	35
blind	0.196	35
ContextEnhancedAdditive	0.193	35
causalGraph	0.182	35
ff	0.19	35
max	0.188	35

Table 9: level 7

translate+preprocess 8.086 s		
name	time (s)	lenght
add	1.5	77
blind	14.8	76
Context-enhanced additive	13.1	76
causal graph	17.47	76
ff	22.0	76
max	26.9	76

Table 10: level 10

translate+preprocess 21.248 s		
name	time (s)	lenght
Additive	0.878	34
blind	2.174	34
ContextEnhancedAdditive	0.717	34
causalGraph	1.024	34
ff	0.755	34
max	1.326	34

6.4 Remarks on general planning and Astro Kid

The Astro kid world can be represented in PDDL, but the domains use of continuous actions, forces the representation to either be relaxed, or made in an awkward way, where the planner have trouble handling it. For PDDL to handle the Astro Kid domain properly, there needs to be introduced a proper way of handling concurrency.

Relaxing the domain ensures that a solution is found quickly, but this solutions is not always valid. This can in some cases be solved by adding noOps at appropriate places. Adding noOps to the the plan when necessary takes a little away form the idea of using the general planner, by requiring some specialised knowledge about the domain, that cant easily be reused on other domains.

Table 11: level 14

translate+preprocess 12.389 s

name	time (s)	lenght
Additive	12.719	273
blind	11.367	158
ContextEnhancedAdditive	104.799	181
causalGraph	20.722	158
ff	33.153	158
max	28.949	158

Table 12: level 32

translate+preprocess 24.389 s

name	time (s)	lenght
Additive	0.934	20
blind	131.198	17
ContextEnhancedAdditive	0.712	17
causalGraph	2.213	17
ff	0.748	17
max	0.961	17

The Update approach guaranties to give a valid optimal solution, due to simulating the complete domain. This approach however have the cost of time and memory used. This is due to the domain not fitting well into the classical domain. The variations of this approach shows this clearly by also hitting into problems. In general the update approach is only useful on domain where the assumption for the classic approach, due to large speed difference an nearly equal quality of the found plans. Even then it only works on simple domains, since it often fails to find a solution due to finite memory and time constraints.

The heuristics causal graph, ff and max have shown to be the most reliable of the available heuristics with no significant difference between them.

7 learning

for an agent to be able solve a problem without prior or very little knowledge, it needs to be able to learn how to handle things it has not seen before. To do this the agent needs experience to learn from, there are various ways of obtaining this experience and how to use it. In the two extreme ends of the spectrum are the two approaches "optimisic" and "pessimistic". which is best depends on the goal of learning and the environment.

Optimistic is build around everything being possible, unless proven otherwise. It focus on exploring the environment by trial and error. This approach works best if all actions in the environment is reversible or it can be reset. This is due to that the agent while exploring the domain inevitable will do something irreversible, (possible resulting in a non solvable level). It is also helpful if the environment can be modified to help exploring hypothesis about the environment.

The approach can be compared to a "breath first search", and will thereby

learn a wide array of possibilities in the domain, this wide knowledge also helps finding solutions of a high quality. however as with BFS it is also slow, especially since time can be wasted on irrelevant knowledge but effective once learned.

an advantage with this approach is that everything is assumed possible unless proven otherwise. this feature is especially advantageous on large domains where only a small subset of objects effects any given action. it also means that it will always find a plan but it can find faulty plans (impossible actions)

independent, dosnt need help from the outside....

Pessimistic this approach is the polar opposite where nothing is excepted to work unless proven otherwise. It is build around only doing thing that is guaranteed to work (already knows how to do), the agent will there for never do something new. The only way of gaining new knowledge is to ask for help from a "teacher" [Walsh and Littman, 2008]. the advantage of this approach is that everything the agent does is guaranteed to turn out as expected, and it will therefore never produce a faulty plan. it can however not guarantee to find a plan without the teacher. "depth first search"

this approach learns quickly, and useful if the environment is irreversible,

use the tutorial traces to speed up the learning

7.1 learning in Astro Kid

It has been shown earlier that to solve problems in the world using fast downward and pddl, it is not feasible to model the complete world, and that it has to be relaxed. Therefore the relaxed version of the domain will be used for the purpose of learning.

to simplify the learning problem.... starting with learning a restricted version of the domain and extending from that. The Action cost feature of PDDL is not considered an will be hard coded to receive consistent results.

this initial restricted version will consist of the basic actions, left right up down push NoOp (no continuous non player movement) this means only parts of the domain are learned and the remaining is left as it is (destroy, gates)....

later it will be extend to commands activate (continuous non player movement)

when looking a the astro kid domain its worth noting that it has a slow learning curve. with tutorials where the most actions are introduced by a trace. using these traces can speed up the learning, since.... these traces is in them self is a good place to start for a pessimistic approach, they however not complete enough to solely base a pessimistic approach on (e.g falling isnt introduced).

7.2 basic learning

the main idea is that if an action is applied on the domain and something happens, what happens must be the effect of that action, and that the action happens means that the preconditions of the actions is fulfilled. It can thereby be concluded that only the literal present in previous state is part of the precondition.....

an action failing on the other hand dosnt give as much information, since it is not known which precondition is violated. however by combining knowledge from several attempts some information can be deduced, the idea is that at least one of the predicate not present is a precondition, the list of possible

candidates can be narrowed down by comparing with those present at a success full action. only when narrowed down to a single candidate can it be guaranteed that candidate is a precondition, that leaves the problem of if the action failed due to multiple precondition violations. if this is the case the best the set of candidate preconditions can due is giving a hint to what the preconditions could be.

an action applied on the domain is considered a success as long as the simulations dosnt reject it. so an action dosnt have to have a visible effect to be applicable. this is due to masked knowledge/effects, this is the case of the effect of the action already being present, and therefore no changes occur

effects of an action are easier to deduce since they are what changes after an action, it is however not necessarily possible top find all effects of an action, due to masked knowledge. this is also the case for disproving effects...

the predicates can be grouped into three groups proven disproven unknown

action succeeds only tells what are not preconditions

failure tells what could be preconditions (missing candidates) candidates tells one situation that dindt work

Algorithm 1 Learning algorithm

```
1: while !goal do
2:     Analyse given trace
3:     make an hypothesis about how the domain works
4:     run planner with hypothesis
5:     get trace from simulation
6: end while
```

its worth mentioning that the complete plan found by the planner might be executable on the domain, but the simulation might still deviate from the predicted result.

when deducing the preconditions ..... typing simplifies grounding of the possible predicates, however in worst case all objects would be of the same type, which would make the typing meaningless, and would therefore have no effect on the algorithm. in a properly typed domain the complexity can often be reduced considerably.

however even with typing, grounding can be a problem since there is no restriction on the number of objects and how they can be relate to each other. In the case of the astro kid world this can clearly be seen with the positions and there relation to each other (map 10x10, 100 fields which can be related 4 different ways (100*2*4*2)). one way to handle this is to assume that only predicates present (with that particular grounding) and the negated equivalent in the problem description are relevant, until proven otherwise by an effect. the idea is to avoid taking various impossible mutations of static objects into consideration. this is especially effective om problems with a underlying fixed structures, such as maps/levels. the negated predicate is need to that .... walk left impossible, then right still possible

this however leaves how to handle "new" predicates not seen before created by effects. when a new predicate is discovered the negated version is add to all candidate preconditions. this however means that a candidate preconditions cant be proven unless all predicates is discovered.

graph representation

to easier detect relations between predicates it is not enough just to know if a given predicate is present, but also what its relation to the other predicates is. A graph representation is therefore used, what is represented is how the objects is related to each other through known predicates.

7.3 generating action schema

candidate Preconditions even though that it is not known which part of the candidate preconditions thats the precondition, its know that the particular combination fails. this can be used when generating the action schema, by adding all the candidate set as preconditions. when the sets are reduced enough the number of candidates can compensate for the lack of precision. this also ensures that no mistake is repeated

effects everything happens unless proven otherwise

7.4 how does it work?

7.5 Conditionals and Disjunction

to further extend the the support for the domain and allow ..... Conditionals and Disjunction needs to be add...

when looking at conditionals and disjunction they have basically the same effect and can be compiled away by splitting the action into multiple actions.

This can done by moving the condition from effects to precondition and treating it as a non conditional action. at the same time adding conditions so only one of the new actions is applicable at any time.

if a conditional have multiple effects they can also be considered separately...

in the extreme case it can be complied down to a single effect per condition p V q -¿ x V z is equivalent to p V q -¿ x V p V q -¿ z

this far down is how ever to be avoid since it means learning the exact same condition twice

using the earlier described approach it is possible to detect conditionals when contradictions happens (a previous actions precondition and effect dosnt match the current).... then the action can then be splited.

the approach of splitting the actions however makes it difficult to merge the knowledge the actions have in common, and in general it becomes more difficult to associate the effect and preconditions of an action to a particular of the given split actions. This is especially the case for actions where the same effect appears from multiple different conditions, or part of the effect is hidden by already present predicates.

the fact that they are split also gives the a problem to learn what the split actions have in common and learn it for all. its only possible to determine when the actions is completely learned, until then only qualified guess is possible

when adding conditionals and disjunctions to the domain it makes it nearly impossible to determine when an action is completely learned. this is due to that every combination of preconditions needs to be tried, unlike earlier where preconditions could be removed on a success. this is also effect the effects.... this means that a new approach is need .... for.... action schemas....

this means that to generalising and effectively to use what has be learned.... it starts to move into fuzzy logic?... which means it can start to produce no plan....

the idea is that since conditionals can add uncertainty (make some experience useless), it can be difficult to deduce meaning from some of the traces... evolutionary algorithms... mmas? using basic approach to manipulate values

avoid storing every single episode since complete knowledge would be need

8 Generalising

one thing is to learn for a specific problem, the knowledge also needs to be transferable to the following levels. This is done by generalising what has been learned

number of objects dosnt matter...

merging conditionals

quantifiers...

8.1 how does it work?

9 Conclusion

9.1 Future Work

it could be interesting to see how the complete Astro Kid domain could be supported by PDDL. the main problem to achieve this is the concurrency created by the continuous actions.

this could be achieve by either extending the language or switching to a multi agent version of PDDL

the idear of switching to a multi agent version fx. MA-PDDL is to threat all objects that can be involved in continuous actions and treating them as independent agents. this could help handling the concurrency issues and thereby allowing the planner to solve levels where this is an issue such as level...

Appendices

A Blocks domain

A.1 Problem Description

```
(define (problem BLOCKS-4-0)
  (:domain BLOCKS)
  (:objects
    D B A C - box
  )
  (:INIT
    (clear C) (clear A) (clear B) (clear D)
    (on C t) (on A t) (on B t) (on D t)
    (handempty))
  (:goal
    (and
      (on D C) (on C B) (on B A)
    )
  )
)
```

A.2 Domain Description

```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:predicates
    (on ?x ?y - object)
    (clear ?x)
    (handempty)
    (holding ?x - box)
  )
  (:types box table - object)
  (:constants t - table)
  (:action pick-up
      :parameters (?x - box ?under - object)
      :precondition (and
      (clear ?x)
      (handempty)
      (on ?x ?under)
    )
      :effect (and
      (not (on ?x ?under))
      (not (clear ?x))
      (clear ?under)
      (not (handempty))
      (holding ?x)
    )
  )
```

```
  (:action put-down
      :parameters (?x - box ?under - object)
      :precondition (holding ?x)
      :effect(and
      (not (holding ?x))
      (clear ?x)
      (not (clear ?under))
      (handempty)
      (on ?x ?under)
    )
  )
)
```

B World Simulaion

Due to the complete set of rules in the Astro Kid world not being be described. The rules are constructed by watching the tutorials and playing the game. This means there most likely are some few inconsistency compared to Astro Kid game itself.

The simulation and editor is created using test driven development and design to follow the paradigm Model View Control

B.1 backend of the simulation

All objects in the world is represented as an object with a set of coordinates defining its position in the 2d grid. Multiple objects can have same position, with the exception of that there can only be one moveable object at any given coordinate set.

Running the simulations is build around a basic input update cycle, where input is the player commands, which can be left, right, up, down or a set coordinates. When a command is given the validity of it is checked and if valid the involved objects are then marked to be move in the given direction. Nothing is actually moved in this stage and if a new command is given before and update the previous one is ignored.

The update consists of all objects are moved to their new positions, when they are moved, they are checked for if they should keep moving next update or stop. objects in the bottom of the grid are updated first to ensure items fall correctly.

B.2 GUI

A basic GUI using Swing is created, it consists mainly of a visual representation of a given Astro Kid world, and some controls for interacting with this world. the world is visualised using a 2D grid of images which is synchronised with the world using a observer observable pattern.

B.3 Solving Problems

The first thing that happens when trying to solve a problem is a PDDL description is generated from the world. This generated description is a direct

representation of the world, so no optimizations are performed on it.

The generated problem description and a predefined domain description is passed to a new instance Fast Downward as command line parameters. The heuristic is also passed this way, as default the FF heuristic is used as it shown to work best on this domain. The output of Fast Downward is piped to a function that extracts the results of the search. The function extracts all actions with a action cost of 1 (which is the player actions) and translates them to Commands that can be executed on the world. Finally each command is executed on a copy of the world, and error massaged appears if the command isn't valid or no solution is found.

C Domain Variation

The original part of the update domain and what its replaced with in the existential approach.

C.1 Original

```
(:action updateDestroy
    :parameters ()
    :precondition (and
    (update updateStage1)
  )
    :effect (and
    (forall (?t - thing ?at ?under - location)
      (when
        (or
          (and
            (boarder ?at)
            (at ?t ?at)
          )
          (and
            (relativ-dir ?at ?under down)
            (at ?t ?at)
            (or
              (and (ground blue ?under) (not (wearing blue)))
              (and (ground purple ?under) (not (wearing purple)))
            )
            (not (exists (?r - robot) (= ?t ?r)))
          )

        )
          (and
            (not (at ?t ?at))
            (not (moving ?t left))
            (not (moving ?t right))
            (not (moving ?t down))
            (clear ?at)
          )
```

```
      )

    )
    (not (update updateStage1))
    (update updateStage2)
  )
)
```

C.2 Changed

```
  (:action updateDestroyBoarder
       :parameters  (?t - thing ?at - location)
       :precondition  (and
              (update updateStage1)
              (boarder ?at)
              (at ?t ?at)
            )
       :effect (and

          (not (at ?t ?at))
          (not (moving ?t left))
          (not (moving ?t right))
          (not (moving ?t down))
          (clear ?at)
        )
  )

  (:action updateDestroyUnder
       :parameters  (?t - thing ?at ?under - location)
       :precondition  (and
              (update updateStage1)
              (relativ-dir ?at ?under down)
              (at ?t ?at)
              (or
                (and (ground blue ?under) (not (wearing blue)))
                (and (ground purple ?under) (not (wearing purple)))
              )
              (not (exists (?r - robot) (= ?t ?r)))
            )
       :effect (and
          (not (at ?t ?at))
          (not (moving ?t left))
          (not (moving ?t right))
          (not (moving ?t down))
          (clear ?at)
        )
  )

  (:action updateDestroy
       :parameters  ()
```

```
        :precondition  (and
                (update updateStage1)
                (not (exists (?t - thing ?at ?under - location)

                  (or
                    (and
                      (boarder ?at)
                      (at ?t ?at)
                    )
                    (and
                      (relativ-dir ?at ?under down)
                      (at ?t ?at)
                      (or
                        (and (ground blue ?under) (not (wearing blue)))
                        (and (ground purple ?under) (not (wearing purple)))
                      )
                      (not (exists (?r - robot) (= ?t ?r)))
                    )

                  )

                  )

                )
              )
        :effect (and

           (not (update updateStage1))
           (update updateStage2)
         )
  )
```

D Levels

insert levels here

D.1 Astro Kid Rules

The player can do the following things: walk left and right, climb up and down on ladders, and push objects horizontally away from it.

References

[Brenner, 2003] Brenner, M. (2003). A multiagent planning language. In *Proc. of the Workshop on PDDL, ICAPS*, volume 3, pages 33–38.

[Garrido et al., 2002] Garrido, A., Fox, M., and Long, D. (2002). A temporal planning system for durative actions of pddl2. 1. In *ECAI*, pages 586–590. Citeseer.

[Walsh and Littman, 2008] Walsh, T. J. and Littman, M. L. (2008). Efficient learning of action schemas and web-service descriptions. *AAAI Conference on Artificial Intelligence*, 23.