A framework of safe robot planning

Roland Pihlakas
August 2008

Abstract
The poster introduces preliminary study for implementing a framework of safe robot planning in Prolog. A principle of safety is introduced which does not depend on explicitly enumerating all possible negative states, but at the same time also does not depend on the robot doing only "what it is told to do". The proposed principle of safety is based on implicit avoidance of irreversible actions, except in explicitly permitted cases.

Introduction

The framework distinguishes between goals and permissions. Therefore the robot will be able to construct its own detailed action plans, given an initial configuration that only contains entries describing the goals and permissions. This part aims at relative independence of the robot.
Secondly, the robot will use only such steps which cause permitted or reversible changes in the environment. Through this principle the robot is expected to be safe: all possible consequences are either permitted (though not obligatory), or are reversible, therefore no harm will be done.

Choice of the programming language

The language for implementation was choosen for the following reasons:

- Prolog has built-in parser available, which will be useful for configuration processing
- Prolog seems to have useful data types, and the data types are optionally of variable type. Among others it supports data types representing numeric ranges and constraints.
- It has conveniently short syntax for some semantics (especially the syntax for failing function calls and resuming the alternatives at some upper levels). Though some syntaxtical structures are also a bit cumbersome (specifically, accessing the fields of record data type), I expect to overcome that quite easily by writing a small preprocessor that allows me to use shorter syntax.
- It has automatic memory management
- It is "scriptable". That means the development can be more interactive, single statements can be tried out at any time, and no long compilations delay my train of thought and the work.

Configuration file

For each dimension of the world (value of some sensory input, or calculated value) it is possible to specify:

- it's set of allowed values (in the future, also a range of values);
- whether it is permitted to change that particular dimension to have any value;
- whether the state of the dimension should be reversed to its original state at the end of the plan;
- its goal value;

- whether it is permitted to change the value of this dimension at all, even temporarily;
- whether it is permitted to change the value of this dimension after asking and getting authorization.

There are two levels of specifications in the configuration. The first level is the obligatory level. The second level is optional level. When the obligatory level fails, the plan fails. When obligatory level succeeds, it is preferable but not mandatory to fulfill also the optional specifications.

An example of the syntax for describing the configuration of goals and permissions:

```
Obligatory {
        Allow x1 = any;
        Free x2;                        // same as Allow any
        Allow x3 = {1, 2, 3};
        Goal y1 = 5;
        Goal y2 = predicate_name;
        Askauth z1;
        Askauth allow z2 = {2, 3};
        Askauth reverse z3;
        Askauth goal z4;        TODO
        Reverse w1;
        Reverse [and] allow w2 = {4, 5};
        Reverse or allow w3 = {9, 10};
        Reverse or goal w4 = 6;
        Dontdisturb q1;
        Dontdisturb or allow q2 = {7, 8};
        Context q3 = 47;
        Dontdisturb [and] allow q4 = 47;                //same as Context = …
        Guarantee for reversing x4 is q5 = 56;
        Guarantee for x5 = 8 is q6 = 99;
}

Obligatory Reverse z5;          //the declarations can be located inside block or be single,
                                //the meaning will stay same

Optional {
        Goal x4 = 2; Cost x3 = 4.0; Scale x3 = *;
        Reverse y3;
}

Budget = 55;
Max steps = 5;
Max time = 10min;
```

Explanation of the entries in the above example
Allow any – any value of the dimension is allowed at all times.
Free – same as allow any.
Allow – allow only given values of the dimension. Must hold at all times during the execution of plan and at the end of the plan.
Goal – the dimension must have specific value or one value from the given set of values at the end of the plan.
Reverse – the dimension must have same value at the end of the plan as it had before beginning of the plan, but it may have any value during the execution of plan.
Askauth – the change is possibly permitted, therefore it should not be discarded from the plan at once, but it needs additional authorization from the user before executing the plan.
TODO: askauth allow, askauth reverse, askauth goal, askauth x4.
Dontdisturb – the dimension's value should not be changed from its original value at any time by the robot. Note that the original value is unspecified in the configuration – instead it is measured before the beginning of each plan. The dimension can have any value. A separate issue for future research is understanding (and optionally permitting instead of reversing) changes that are caused by external agents or by natural causes.
Context – specifies dimension's value that must hold at all times, including before the start of the plan.
Guarantee for – having / creating certain value in one dimension is a guarantee that it is possible to achieve certain value in other dimension.
Guarantee for reversing – having / creating certain value in one dimension is a guarantee for reversing the value of the other dimension to it's original value.
Guarantee for nodisturb - TODO
Reverse or allow – either the value must be reversed or have specified value. Must hold at all times during the execution of the plan and at the end of the plan.
Reverse [and] allow – the value must be reversed to its unspecified original value at the end of the plan, but it can have only specified values during the execution and at the end of the plan.
Reverse or goal – the dimension may have any value during the execution of plan, but must have its original value or one given value at the time the plan ends.
Dontdisturb or allow – the value must either be not changed from its original or must have a specified value.
Dontdisturb [and] allow – same as Context.
Budget, **Cost** and **Scale** – explained below in the "Cost" paragraph
Max steps – maximum search depth, relevant to finding more optimal solutions
Max time - TODO

The Backus–Naur Form of the configuration is roughly as follows:
<configuration> ::= <block>+
<block> ::= <goals_block> | <budget> | <maxsteps> | <maxtime>
<budget> ::= "Budget" "=" <value> ";"
<maxsteps> ::= "Max steps" "=" <value> ";"
<maxtime> ::= "Max time" "=" <time_value> ";"
<time_value> ::= ….. **TODO**
<goals_block> ::= <big_block> | <small_block>

<big block> ::= <blocktype> “{“ <entries> “}”
<small_block> ::= <blocktype> <entry>
<blocktype> ::= “Obligatory” | “Optional”
<entries> ::= <entry>*
<entry> ::= <entrytype> <dimension_name> [“=” <values_field>] “;”
<entrytype> ::= “Allow” | “Free” | “Goal” | “Askauth” | “Reverse” | “Dontdisturb” | “Context” | “Dontdisturb or allow” | “Reverse or allow” | “Reverse or goal” | **TODO**
<values_field> ::= <predicate_name> | “any” | <values> | <value>
<values> ::= “{“ <value> { “,” <value> } “}”
<comment> ::= <singleline_comment> | <multiline_comment>
<singleline_comment> ::= “//” <text> <EOL>
<multiline_comment> ::= “/*” <text> “*/”
The whitespaces and comments are discarded before processing the configuration, and the configuration is case-insensitive.

The configuration in the above format will be translated to three data structures describing the conditions that will be checked for:
1) before the start of plan – preconditions;
2) after every step of the plan – keep-always conditions;
3) at the completion of final step of the plan – goal conditions.

The data structures will contain more concrete values for every dimension to be checked. All three data structures have two versions: obligatory and optional.

Follows the table that describes how the entries in configuration will be translated to the three data structures.

	Preconditions	Keep-always conditions	Goal conditions
Allow any / free (the default for optional goals)	*	*	*
Allow values	*	{values}	{values}
Goal	*	*	{values}
Reverse (the default for obligatory goals)	*	*	original value
Dontdisturb	*	original value	original value
Askauth **TODO: more**	*	* if authorized; else original value	* if authorized; else original value
Context (dontdisturb allow)	{values}	original value	original value
Reverse allow	{values}	{values}	original value
Dontdisturb or allow	*	original value or {values}	original value or {values}
Reverse or allow	*	{values} **TODO:** – **should be either ***	original value or {values}

		– **or "original value or {values}" (and NB! – this duplicates the Dontisturb or allow type)**	
Reverse or goal	*	*	original value or {values}

Legend:
"*" – any value
"{values}" – set of values or one value specified in the respective field of configuration. If any of the values in the set matches dimension's actual value, the condition is met. The concrete values can be replaced by a more complex predicate that must be true for the dimension's actual value in order for the condition to be met.
"original value" – the value of the dimension that was measured before the beginning of the plan.

The "Reverse" condition is the default condition for all dimensions for which no corresponding entries exist in the "obligatory" block of the configuration.
The "Allow any" condition is the default condition for all dimensions for which no corresponding entries exist in the "optional" block of the configuration.

Dealing with uncertainties
1) When an action or other causal relation results in unknown value of some dimension, the action is permitted only if the affected dimension is allowed to have any value, else the plan fails.
2) When an action or other causal relation has different possible result values in some dimension, and **any** of these values contradicts the configuration, the plan fails. That is, the worst is assumed.
3) Similarly, when any of the multiple possible results of some causal relation means the budget will be exceeded, the plan fails. Again, the worst is assumed.
(2) and (3) above mean that the planner works according to the Minimax rule: it tries to maximise its the worst-case outcomes – that is, during the planning it assumes that worst will happen when given chance to.

Cost entries
Each action can have a cost associated with it **(TODO: syntax for that?).** Also the world's dimensions can have cost associated with them so that the cost depends on the discrepancy between allowed / preferred value and actual value. Therefore it will be possible to calculate the cost of the plan. Plans that deviate more from preferred states will be more costly. Because deviating from goals also has a cost, not following the optional plans at all may be more costly than fulfulling them. The robot has a budget which the total cost of the plan must fit, else the plan fails. Obligatory entries have usually infinite cost associated, so that failing them means failing the plan. Optional

entries have some countable **(TODO: is this the correct word?)** cost associated **(by default: TODO).**
It is sometimes meaningful to associate obligatory entries also with countable costs **(TODO: more examples)** instead of the default infinite costs. An example is playing some two-player game where playing is always preferred over not playing, but playing to 0-0 result is always preferred to losing. Therefore the first goal is to guarantee at least the 0-0 result and optional goal is to make such moves that give the opponent more opportunity to make mistakes.

General structure of the program
The module for parsing the configuration and for safe planning is separate from the module that learns or calculates consequences of actions and external events. The latter can be some fixed function or module that is able to learn. The only **(TODO: more requirements?)** requirement for the predicting module is that when it is not able to predict some dimension with certainty, it either should return the set of all **(correctly/accurately ?)** expected values, or return special value indicating “any value is possible”. The latter value may be returned irrespective to its truthfulness. The meaning of "any value is possible” is that it allows the planning algorithm to expect the worst case that is possible according to the configuration.

Protocol for writing the configuration
Often the robot is competent only in certain work areas or work places. By competence I mean here mainly that its prediction module is able to do accurate predictions.
By default the robot does not do anything.
When an goal is given, the robot may start doing something when it either knows that it is able to reach the goal without disturbing other dimensions, or it’s prediction module is badly written or badly trained and does not announce its possible incompetence. **(TODO: example for bad training that turns off the return of warning values).**
Assuming that the prediction module is satisfactory, usually some additional permissions (Allow entries) should be written in order for the robot to achieve previously given goals. Permissions can easily be too general and such robot may be unsafe in some situations where the goals seem to the robot to be achievable given the permissions, but actually there are surprises waiting “around the corner”. Such surprises will be likely because in other contexts different causal rules might apply, but the robot at the same time might be quite certain, based on its previous experience, that its predictions are correct. Therefore Context entries must be added.
A partial solution would be automatically generating context entries either at the time the permissions / rights are given, or at the time of training.

Conclusion
It seems that the set of alternative configuration entries for each dimension is quite big and it is likely to become even bigger in the future.
An other confusing part (for the user) is the issue that when Goals or Rights entries (these are Allow entries) are written, usually corresponding Context entries should be added. But I find that it is yet too easy to miss doing the latter properly. An other thing that has to be improved is finding more easily comprehensible names for different kinds of rights.

References

The principle of avoiding irreversibilities is taken from [3], [4], [5] and can be found also in [2]. But both sources use explicit avoidance of irreversibilities. I would like to propose and stress the importance of implicit avoidance of irreversibilities.
The concept of Guarantees is taken from [2].
The distinction between Permissions and Authorisation asking is taken from [1].

[1] J. Fox, S. Das, *Safe and sound: Artificial Intelligence in Hazardous Applications*. London: AAAI Press / The MIT Press, 2000

[2] D. Weld, O. Etzioni, The First Law of Robotics (a call to arms), *AAAI-94, 1994*. URL: http://www.cs.washington.edu/homes/etzioni/papers/first-law-aaai94.pdf

[3] A. Eppendahl, M. Kruusmaa., "Obstacle Avoidance as a Consequence of Suppressing Irreversible Actions", *Proceedings of EpiRob*, 2006. URL: http://homepage.mac.com/a.eppendahl/work/papers/Eppendahl-obsacs.pdf

[4] A. Eppendahl, M. Kruusmaa, Y. Gavshin, "Don't Do Things You Can't Undo: Reversibility Models for Generating Safe Behaviours", 2007. URL: http://homepage.mac.com/a.eppendahl/work/papers/Eppendahl-dondty.pdf

[5] J. Gavšin, "Using the Concept of Reversibility to Develop Safe Behaviours in Robotics" (dissertation). Tartu: Tartu Ülikooli Arvutiteaduse instituut, 2007. URL: http://hdl.handle.net/10062/1990

Errata

This is a work in progress presented for communication purposes. This is not a comprehensive guide to healing the lives or the world, so I'm not responsible if You mess something up, or fail to prevent a mess, or the when meteorite has already fallen onto your just-cleaned-up garden.