1 ngames/evaluation/interpreter.pl

query(*?Term:term*) *[det]*

Substitute for the usual `call/1` predicate, but includes support for terms expressed as conjunctions (e.g. A and B).

Arguments

Term	A Prolog term to be queried.

query_rule(*?Rule:term*) *[det]*

Return True if *Rule* is active given the current state of the system.

Arguments

Rule	A '`rule/4`' predicate.

find_consequences(*+ID:atom, +Type:atom, +Threshold:int, -L:list*) *[det]*

Find the instantitations of the if-then-where rules of the '*Type*' kind that are currently active, and extract their consequences paired with their priority. Active rules with priority larger than '*Threshold*' are excluded.

Arguments

ID	Identifier for the action situation.
Type	One of either 'boundary', 'position', 'choice', 'control' or 'payoff'.
Threshold	The maximum priority of rules to be considered.
L	The output list with the processed consequences.

delete_key_gt(*L:list, N:int, NewL:list*) *[det]*

Auxiliary predicate to delete consequences whose priority is over some given threshold.

Arguments

L	List with priority-fluents pairs to be processed.
N	Threshold, fluents with priority over it are to be excluded.
NewL	List identical to '*L*', but fluents with priority over Threshold are excluded.

process_consequences(*+Conseqs:list, +OldParts:list, ?NewParts:list*) *[det]*

Get all the consequences of some rule type and process them in decreasing order of priority. It returns a new list indicating the consequences that hold, negated ones (aka overwritten) included. The list of consequences is processed in a way such that consequences that are in conflict with consequences of higher priority are discarded.

It should be called as:

```
?- find_consequences(boundary,L),process_consequences(L,[],P).
```

Arguments

Conseqs	List of consequence of some rule type.
OldParts	List of old consequences. Intended to be called with the empty list.
NewParts	List of the consequences that hold, negations included.

get_simple_consequences(*+ID:str, +Type:str, +Threshold:int, -L:list*) *[det]*

Process the consequences of boundary, position, choice and payoff rules. It gets the consequences of the rules with the given identification and type, has rules of higher priority overwrite

rules of lower priority, and finally deletes negated (aka overwritten) facts. It returns the result in a list of fluents.

Arguments	
ID	Identifier of the action situation.
Type	One of either 'boundary', 'position', 'choice' or 'payoff'.
Threshold	Consequences of rules with priorities exceeding '*Threshold*' are excluded.

control_conseq_fact_incompatible(+*Fact:atom, +S:list*) *[det]*

Check whether a single fact is compatible with a list of established facts.

Arguments	
Fact	The fluent whose compatibility we want to check.
S	List of previously established facts.

control_conseq_incompatible(+*Facts:atom, +S:list*) *[det]*

Checks whether the fluents in '*Facts*' that make up a joint consequence statement of an active control rule are incompatible with the next states already derived in '*S*'.

Arguments	
Facts	*Facts* derived from a new control rule. Either a single fact or a conjunction of them (aka 'A and B').
S	List of potential next state descriptions already derived (aka a list of lists).

control_rule_incompatible(+*Conseqs:list, +S:list*) *[det]*

Check whether the '*Conseqs*' list of an active control rule is compatible with the next states already derived in '*S*'.

Arguments	
Conseqs	List of facts derived from a control rule. Each of them is either a single fact or a conjunction (aka 'A and B').
S	List of potential next state descriptions already derived (aka a list of lists).