How to Prepare New Domains

Domain Examples

There are example domains in *benchmarks* folder.

Domain Construction Hints (with Colorballs domain example)

Facts

- These are never altered by any of sensing or actuation actions and does not need any time parameter.

Code

`ball(B)` : ball id
`color(C)` : color id
`bin(Bi)` : bin id
`point(R,C)` : point id consists of row(R) and col(C)
`location(point(R,C))` : location identified by point(R,C)
`bin_location(B,L)` : bin(Bi) location relation with location(L)
`bin_color(B,C)` : bin(Bi) color relation with color(C)
`adj(L1,L2)` : location(L1) adjacency relation with location(L2)

```
ball(0..number_of_balls-1).

%All colors a ball or trash can have
color(red;blue;purple;green).
#const color_size=4.

%All bins
bin(t1;t2;t3;t4).

%All locations on map
location(0..loc_size-1).

%Relationship between bins and locations
bin_location(t1,0).
bin_location(t2,map_col_size-1).
bin_location(t3,loc_size-map_col_size).
bin_location(t4,loc_size-1).

%Relationship between bins and colors
bin_color(t1,red).
bin_color(t2,blue).
bin_color(t3,purple).
bin_color(t4,green).

%Relationships between locations
%Note that a location is not adjacent to itself
adj(L,L+map_col_size):-location(L),L+map_col_size<loc_size.
adj(L,L+1):- location(L), (L+1)\map_col_size!=0.
adj(L1,L2) :- adj(L2,L1), location(L1), location(L2).
```

Fully Observable Fluents

- Fully observable fluents are only effected by actuation actions and can be determined by checking the 'effect' title of actuation actions will reveal them.
- Since their values depend on time, they need time as parameter.
- All fluents should have negative correspondent (except the functional fluent that preserve uniqueness and existence)
- For each fluent consider: Inertia, Default,Uniqueness and Existence
  - Inertial: All Fluents are Inertial since problems we are dealing occur in static and deterministic environments.
  - Uniqueness: If predicate has single instance at initial state and at the 'effect' of each 'action' if one introduced then another one is removed then the predicate has Uniqueness.
  - Existence: If predicate is introduced at initial state and at the 'effect' of each 'action' if one is removed at least one another is introduced then the predicate has Existence.
  - Default: If predicate is not introduced at initial state it should has negative corresponding as default.

Code

`at(L,T)` : location(L) of agent at time(T).

- **Requires INERTIA** if agent is at a place, without any actuation action it will remain there.
- **Requires UNIQUENESS and EXISTANCE (FUNCTIONAL)** The agent must be in and only a location at any time.

```
%Intertia
{at(L,T+1)} :- at(L,T), location(L), atime(T).
%Uniqueness
:- 2{at(L,T): location(L)}, time(T).
%Existance
:- {at(L,T): location(L)}0, time(T).
```

`trashed(B,T)` : if ball(B) is trashed into bin(Bi) at time(T).

- **Requires DEFAULT** for being not trashed for each ball, problem states that no ball is trashed at the beginning.
- **Requires INERTIA** if ball(B) is trashed in bin(Bi); without any actuation action it will remain there.
- **Requires UNIQUENESS** in bin(Bi); each ball can be trashed only to one of the bins.

```
%Default
-trashed(B,time_min) :- not trashed(B,time_min) , ball(B).
%Intertia
trashed(B,T+1) :- not -trashed(B,T+1), trashed(B,T), ball(B), atime(T).
-trashed(B,T+1) :- not trashed(B,T+1), -trashed(B,T), ball(B), atime(T).
```

`holding(B,T)` : if ball(B) is being hold by agent at time(T).

- **Requires DEFAULT** for being not holding; problem states nothing is being hold at the beginning.
- **Requires INERTIA** If ball(B) is hold, without any actuation action it will remain there.
- **Requires UNIQUENESS** in ball(B) problem states agent cannot hold more than one ball.

```
%Default
-holding(B,time_min) :- not holding(B,time_min), ball(B).
%Inertia
holding(B,T+1)  :- not -holding(B,T+1),  holding(B,T),  ball(B), atime(T).
-holding(B,T+1) :-  not holding(B,T+1), -holding(B,T), ball(B), atime(T).
```

Partially Observable Fluents

- Partially observable fluents (POF) may effected by both actuation and sensing actions, and can be determined by checking the 'effect' title of sensing actions will reveal them.
- Since their values depend on time, they need time parameter.
- POF should be revealable by sensing actions.
- Each POF should have external predicate correspondent.
- Two types of POF: Boolean, Multivalue
  - **Boolean**: Fluent should have negative correspondent.
  - **Multivalent**: Fluent does not need to have negative correspondent.
- For each fluent consider: Inertia, Default,Uniqueness and Existence
  - **Inertial**: All Fluents are Inertial since problems we are dealing occur in static and deterministic environments.
  - **Uniqueness**: If predicate has single instance at initial state and at the 'effect' of each 'action' if one introduced then another one is removed then the predicate has Uniqueness.
  - **Existence**: POF does not have Existence property. They may be unknown at any time during plan. So if all other options are eliminated, the remaining one is true.
  - **Default**: POF **should not have** Default property. They can be unknown at any thime and no value should be automatically assigned in this case.

Knowledge Types of boolean POF

Known Positive po_fluent()	We know po_fluent is TRUE	can be proven by sensing action
Known Negative -po_fluent()	We know po_fluent is FALSE	can be proven by sensing action
Unknown Positive not po_fluent()	We DON'T know po_fluent is TRUE	state when sensing action not occurred
Unknown Negative not -po_fluent()	We DON'T know po_fluent is FALSE	state when sensing action not occurred

Knowledge Types of multivalent POF

We can also define Partially Observable Fluents with Values.

Known Positive po_fluent(V)	We know po_fluent is TRUE	can be proven by sensing action ,sensing action will directly return the TRUE fluent
Known Negative -po_fluent()	We know po_fluent is FALSE	will be reached after we find TRUE value by sensing action.
Unknown Positive not po_fluent()	We DON'T know po_fluent is TRUE	state when sensing action not occurred
Unknown Negative not -po_fluent()	We DON'T know po_fluent is FALSE	state when sensing action not occurred

Code

`ball_at(B,L,T)` : ball(B) is at location(L) at time(T).

- **Requires INERTIA** if ball is at a place, without any actuation action it will remain there.
- **Requires UNIQUENESS** Ball can be in only one place.
- **Requires EXTERNAL PREDICATE**`sensed(ball_at(B,L),T)___`

```
%Intertia
ball_at(B,L,T+1) :- not -ball_at(B,L,T+1), ball_at(B,L,T), ball(B), location(L), atime(T).
-ball_at(B,L,T+1) :- not ball_at(B,L,T+1), -ball_at(B,L,T), ball(B), location(L), atime(T).
%Uniqueness
:-2{ball_at(B,L,T):location(L)},ball(B),time(T).
```

`ball_color(B,C,T)` : ball(B) has color(C) at time(T).

- **Requires INERTIA** if ball is at a place, without any actuation action it will will remain there.
- **Requires UNIQUENESS** Ball can be in only one place.
- **Requires EXTERNAL PREDICATE** `sensed(ball_color(B,C),T)`

```
%Intertia
ball_color(B,C,T+1) :- not -ball_color(B,C,T+1), ball_color(B,C,T), ball(B), color(C), atime(T).
-ball_color(B,C,T+1) :- not ball_color(B,C,T+1), -ball_color(B,C,T), ball(B), color(C), atime(T).
%Uniqueness
:-2{ball_color(B,C,T):color(C)},ball(B),time(T).
```

Actuation Actions

- Actuation actions are **EXOGENEOUS**.
- For each followings should be considered
- Preconditions: These will be represented by constraints that prevent an action to occurre
- Effects : Results of actuation actions

Code

`move(L,T)` : move to location(L) at time(T)

- **Precondition** agent cannot move to a place which is not adjacent to current location(L)
- **Effect** agent will be at(L) at time(T+1).

```
%Exogeneous
{move(L,T)} :- location(L), atime(T).
%Precondition
:- move(L2,T), at(L1,T), not adj(L1,L2), location(L1), location(L2), time(T).
%Effect
at(L,T+1) :- move(L,T), location(L), atime(T).
```

`pickup(B,T)` : pickup ball(B) at time(T)

- **Precondition** agent cannot pickup ball(B) if ball is not in same location as agent
- **Effect** agent will be holding ball(B)

```
%Exogeneous
{pickup(B,T)} :- ball(B), atime(T).
%Precondition
%"ball_at" is partially observable fluent, so we need to KNOW if ball_at(B,L,T)
:- pickup(B,T), at(L,T), not ball_at(B,L,T), ball(B), location(L), time(T).
%Effect
holding(B,T+1) :- pickup(B,T), ball(B), atime(T).
```

`place(Bi,T)` : place a ball(B) to bin(Bi) at time(T)

- **Precondition** agent must holding ball(B)
- **Precondition** agent must be at a location(L) contains a bin(Bi)
- **Precondition** Color of holding ball and bin must match
- **Effect** ball(B) will be trashed into bin(Bi)
- **Effect** not holding that ball any more.

```
%Exogeneous
{place(Bi,T)} :- bin(Bi), atime(T).
%Precondition
:- place(Bi,T), {holding(B,T): ball(B)}0, bin(Bi), time(T).
%Precondition
:- place(Bi,T), bin_location(Bi,L1), not at(L2,T), time(T).
%Precondition
holding_color(C,T):- holding(B,T), ball_color(B,C,T), ball(B), color(C), time(T).
:- place(Bi,T), bin_color(Bi,C), not holding_color(C,T), time(T).
%Effect
trashed(B,T+1) :- place(Bi,T), holding(B,T), bin(Bi), ball(B), atime(T).
%Effect
-holding(B,T) :- place(Bi,T), holding(B,T), bin(Bi), ball(B), atime(T).
```

Sensing Actions

- Sensing actions are exogeneous.
- Sensing actions **must only effect external predicates**.
- For each followings should be considered
- Preconditions: These will be represented by constraints that prevent an action to occurre
- Effects : Indicates all possible outcomes.

Code

`sense(ball_at(B,L),T)` : Sense if ball_at(B,L,T) as ball(B), location(L) at time(T), is TRUE or FALSE

- **Precondition** ball location (ball_at) must be Unknown
- **Precondition** agent must be at location(L)
- **Effect** external predicate `sensed(ball_at(B,L),T)` or `sensed(-ball_at(B,L),T)` becomes TRUE

```
%Exogeneous
{sense(ball_at(B,L),T)} :- ball(B),location(L),atime(T).
%Precondition
:- sense(ball_at(B,L),T), 1{ball_at(B,L,T);-ball_at(B,L,T)}, ball(B), location(L), time(T).
%Precondition
:- sense(ball_at(B,L),T), at(L1,T), L!=L1, ball(B), location(L), location(L1), time(T).
%Effect
1{sensed(ball_at(B,L),T+1);sensed(-ball_at(B,L),T+1)}1 :- sense(ball_at(B,L),T), ball(B), location(L), ati
```

`sense(ball_color(B),T)` : Sense the balls color

- **Precondition** ball color (ball_color) must be Unknown
- **Precondition** agent must be holding ball(B)
- **Effect** external predicate sensed(ball_color(B,C),T) will be TRUE

```
%Exogeneous
{sense(ball_color(B),T)} :- ball(B),atime(T).
%Precondition
:- sense(ball_color(B),T), 1{ball_color(B,C,T):color(C)}, ball(B), time(T).
%Precondition
:- sense(ball_color(B),T), not holding(B,T), ball(B), time(T).
%Effect
1{sensed(ball_color(B,C),T+1):color(C)}1 :- sense(ball_color(B),T), ball(B), atime(T).
```

State Constraints

As in unconditional problems.

Ramifications

There are 3 ramifications must be introduced to domain for Partially Observable Fluents (POF):

- Result of sensing action (which is can be an external predicate) should cause an effect on POF.
- Among all possible results of sensing action only one of the can be true for a plan. By saying that we can conclude that if one of the POF indicating a property is true, then all other possible outcomes are accepted to be wrong for that particular plan.
- If it is proved that all outcomes for a sensing action be wrong but one, then remaining one is accepted to be true, because of the assumptions of problem and definition of sensing action in other benchmarks.

Code

Direct Effect of External Predicate

```
ball_at(B,L,t+time_min) :- sensed(ball_at(B,L),t+time_min), ball(B),location(L).
-ball_at(B,L,t+time_min) :- sensed(-ball_at(B,L),t+time_min), ball(B), location(L).

ball_color(B,C,t+time_min) :- sensed(ball_color(B,C),t+time_min), ball(B), color(C).
-ball_color(B,C,t+time_min) :- sensed(-ball_color(B,C),t+time_min), ball(B), color(C).
```

Uniqueness of Partially Observable Fluents

```
%A ball can be only one place
-ball_at(B,L,t+time_min) :- ball_at(B,L1,t+time_min), ball(B), location(L), location(L1), L!=L1.

%A ball can have only one color
-ball_color(B,C,t+time_min) :- ball_color(B,C1,t+time_min), ball(B), color(C), color(C1), C!=C1.
```

Existance of Partially Observable Fluents

```
%If all other options are eliminated, the remaining one is true.
ball_at(B,L,t+time_min) :-
      loc_size-1{-ball_at(B,L1,t+time_min):location(L1),L1!=L}loc_size-1,
      ball(B), location(L).

%If all other options are eliminated, the remaining one is true.
ball_color(B,C,t+time_min) :-
      color_size-1{-ball_color(B,C1,t+time_min):color(C1),C1!=C}color_size-1,
      ball(B), color(C).
```

Concurrentcy Constraints

- Non-Concurrency is necessary for two reasons:
- Problem domain prevents actions to occur simultaneously, for example an agent may not move and pickup an object at the same time, if pickup action require a still position to succeed.
- **HCP-ASP requires**:
  - There can be only one sensing action at a time.
  - A sensing action and an actuation action cannot occur simultaneously.
  - Multiple actuation actions are allowed to occur at the same time, as long as it is not restricted by domain.

Additional Note on State Fluents

HCP-ASP tool tends to eliminate duplicate computations. One of the mechanisms is such that: A conditional plan is a collection individual plans that form a single plan deviating in sensing actions which means they are sharing a certain part of the plan with other possible branches. Instead of giving classical planner the *ultimate initial state* for problem and asking a plan for possible world states (each possible outcome reflects different world state); we are generating a new planning problem starting from the branching point (sensing action) and asking for a plan. Then classical planner generates a shorter plan (considering NP complexity of problem, reducing problem size is a mandatory point) which we attach to the previous branch where world state is diverted from. That is why determining and propagating fluents are very important.

But HCP-ASP does not force you for the behavior explained above. While calling the binary if you add `--plan_from_beginning` flag, tool will calculate each plan from *problems initial state* rather than intermediate artificial *initial states*. This is mandatory if your **domain has constraints and optimizations regarding**:

- plan size
- past actions or states

which requires complete evaluation of each branch from problems initial state to goal.

Although with simple tricks you can still construct a domain that benefits of computational cutbacks by: Include the necessary information for points mentioned above as state fluents and make sure they are updated and propagated in necessary contexts.

Example

- You have a constraint regarding plan size.
  - Include `time` or `step` function to fluents in DIF file.
- You have a constraint regarding number of sensing (or actuation) action
  - Introduce a new **functional** fluent which will increase with occurrence of action and propagate in time. Since this fluent will be a functional fluent (will hold uniqueness and existance)
  - Then your constraints should be on the fluent (`no_actions(A,T)`).

Code

```
%Initial
no_actions(0,0).

%Intertia
{no_actions(A,T+1)} :- no_actions(A,T), actions(A), atime(T).
%Uniqueness
:- 2{no_actions(A,T): actions(A)}, time(T).
%Existance
:- {no_actions(A,T): actions(A)}0, time(T).

%Direct effect of actions
no_actions(A+N,T+1) :-
    #count{action_A(P1,P2,T):param(P1),param(P2);action_B(P3,T):param(P3)},
    no_actions(A,T), time(T).
```

- You have a constraint regarding if an action_A happened before, action_B cannot be performed.
  - Keep a **functional fluent** indicating occurrence of action_A.

Initial State File

- Initial state file should list only state fluents representing initial state.

- These fluents will be converted to constraints during planning such as

```
at(0,0).
holding(Ball_1,0).
```

  will be included in problem file as

```
:- not at(0,0).
:- not holding(Ball_1,0).
```

- **No other file should contain this information**.

Domain Rules

- No domain or problem file should include **initial state**. It will be stated in a separate file, listing initial state fluents.
- Please see *Concurrency Constraints* above.

Certain modes require additional rules

Default Mode

- Domains should define time-step limitation with constants.

```
time(time_min..time_max).
atime(time_min..time_max-1).
```

Incremental Mode

- All time usages under `#program base.` must use `time_min` instead of `0`.

```
%Default
-holding(B,time_min) :- not holding(B,time_min), ball(B).
```

- Exogeneous actions must be also mentioned under `#program base.` since they can occur at time `time_min`.

```
{move(L,time_min)} :- location(L).
{pickup(B,time_min)} :- ball(B).
{place(B,time_min)} :- ball(B).
```

- Rules under `#program step(t).` and `#program check(t).` must indicate time with `t+time_min`. In another saying replace `t` with `t+time_min`.

```
%Inertia
someFluent(B,t+time_min) :- not -someFluent(B,t+time_min),
    someFluent(B,t+time_min-1), forSome(B).

%Direct Effect
someFluent(L,t+time_min) :- someAction(L,t+time_min-1), forSome(L).
```

- Query in `#program check(t).` should still use time `t` for query.

```
%Query for some goal
:- query(t),not goal(t+time_min).
```

Domain Insight File (DIF)

An XML file contains necessary information to process ASP solver outcome. Starts with Root Element `<hcp>...</hcp>`

The contents obey mostly Clingo ASP syntax such as: * constants start with lowercase letter * Variables start with Uppercase letter * Usage of literals in functions are allowed. * Nested functions. * Variables are recognized which means if you use same variable name in a scope, in order function to be recognize they must have same value in ASP output. * (!) No mathematical operators are recognized.

There is no restriction in filename or extension. (!) Note that DIF file is problem specific since it contains the **initial state**. Each problem instance should have its own *DIF* file.

Time Variable Indicator `time_variable`

Declares the variable through out DIF. Not necessarily match the usage in ASP formalization.

```
<time_variable name="T" />
```

Time Predicate `time_predicate`

Indicates the time function used in ASP formalization.

(!) The variable indicating time value must match with *Time Variable Indicator*.

```
<time_predicate function="time(T)" />
```

Goal Predicate `goal_predicate`

Indicates the goal function used in ASP formalization.

(!) The variable indicating time value must match with *Time Variable Indicator*.

```
<goal_predicate function="goal(T)" />
```

Min-Max Time Sonstants `clingo_time_min_var,clingo_time_min_var`

Indicates the time limit constants used in domain.

```
<clingo_time_min_var name="time_min" />
<clingo_time_max_var name="time_max" />
```

State Fluents `<state_fluents> [<fluent function="..."/>]+ </state_fluents>`

- Indicates State Fluent functions used in domain.
- Constants arguments, nested functions and negative functions are allowed.
- Fluents **must** have time parameter. (!) The variable indicating time value must match with *Time Variable Indicator*.

```
<state_fluents>
    <fluent function="at(L,T)"/>

    <fluent function="holding(B,T)"/>
    <fluent function="-holding(B,T)"/>
```

```
    <fluent function="carrying(withTray,O,T)"/>
    <fluent function="carrying(withByHand,O,T)"/>

    <fluent function="crossedLine(at(A,T),at(B,T)"/>
    ...
</state_fluents>
```

Actuation Actions `<actuation_actions> [<action function="..."/>]+ </actuation_actions>`

- Indicates Actuation action functions used in domain.
- Constants arguments, nested functions and negative functions are allowed.
- Variables in *action* declaration are recognized which means if you use same variable, in order function to be recognize they must have same value in ASP output.
- Actions **must** have time parameter. (!) The variable indicating time value must match with *Time Variable Indicator*.

```
<actuation_actions>
    <action function="pickup(O,T)"/>
    <action function="place(B,T)"/>
    <action function="move(loc(X1,Y1),loc(X2,Y2),T)"/>
    ...
</actuation_actions>
```

Sensing Actions `<sensing_actions> [<action function="..."> [<outcome function="..."/>]+ </action>]+ </sensing_actions>`

- Indicates Sensing action and Outcome functions used in domain.
- Constants arguments, nested functions and negative functions are allowed.
- Variables in *action* declaration are recognized which means if you use same variable, in order function to be recognize they must have same value in ASP output.
- Actions and outcomes **must** have time parameter. (!) The variable indicating time value must match with *Time Variable Indicator*.

```
<sensing_actions>
    <action function="sense(ball_at(B,L),T)">
        <outcome function="sensed(ball_at(B,L),T)"/>
        <outcome function="sensed(-ball_at(B,L),T)"/>
        ...
    </action>

    <action function="sense(ball_color(B),T)">
        <outcome function="sensed(ball_color(B,red),T)"/>
        <outcome function="sensed(ball_color(B,green),T)"/>
        <outcome function="sensed(ball_color(B,blue),T)"/>
    </action>

    <action function="sense(location(L),if(S),then(H),T)">
        <outcome function="differentSensed(someF(L,T)"/>
        <outcome function="differentSensed(someOtherF(H,T)"/>
        <outcome function="differentSensed(someOtherOtherF(L,S,T)"/>
    </action>
```

```
    ...
</sensing_actions>
```
