1 BNF definition of PDDL 3.1

Hereby a complete BNF syntax definition of the PDDL 3.1 language is presented (with comments and indications of minor corrections) based on the originally published articles and information about PDDL 1.2, 2.1, 2.2, 3.0 and 3.1 [1-5].

1.1 Domain description

```
<domain>                           ::= (define (domain <name>)
                                           [<require-def>]
                                           [<types-def>]:typing
                                           [<constants-def>]
                                           [<predicates-def>]
                                           [<functions-def>]:fluents
                                           [<constraints>]
                                           <structure-def>*)
<require-def>                      ::= (:requirements <require-key>+)
<require-key>                      ::= See Section 1.3
<types-def>                        ::= (:types <typed list (name)>)
<constants-def>                    ::= (:constants <typed list (name)>)
<predicates-def>                   ::= (:predicates <atomic formula skeleton>+)
<atomic formula skeleton>          ::= (<predicate> <typed list (variable)>)
<predicate>                        ::= <name>
<variable>                         ::= ?<name>
<atomic function skeleton>         ::= (<function-symbol> <typed list (variable)>)
<function-symbol>                  ::= <name>
<functions-def>                    ::=:fluents (:functions <function typed list (atomic function skeleton)>)
<function typed list (x)>          ::= x+ - <function type> <function typed list(x)>
<function typed list (x)>          ::=
<function typed list (x)>          ::=:numeric-fluents x+
                                       This is deprecated since PDDL 3.1, where the default fluent type is number.
<function type>                    ::=:numeric-fluents number
<function type>                    ::=:object-fluents object
<function type>                    ::=:typing + :object-fluents <type>
<constraints>                      ::=:constraints (:constraints <con-GD>)
<structure-def>                    ::= <action-def>
<structure-def>                    ::=:durative-actions <durative-action-def>
<structure-def>                    ::=:derived-predicates <derived-def>
<typed list (x)>                   ::= x*
<typed list (x)>                   ::=:typing x+ - <type> <typed list(x)>
<primitive-type>                   ::= <name>
<primitive-type>                   ::= object
<type>                             ::= (either <primitive-type>+)
<type>                             ::= <primitive-type>
<emptyOr (x)>                      ::= ()
<emptyOr (x)>                      ::= x
<action-def>                       ::= (:action <action-symbol>
                                           :parameters (<typed list (variable)>)
                                           <action-def body>)
<action-symbol>                    ::= <name>
<action-def body>                  ::= [:precondition <emptyOr (pre-GD)>]
                                       [:effect <emptyOr (effect)>]
<pre-GD>                           ::= <pref-GD>
<pre-GD>                           ::= (and <pre-GD>*)
<pre-GD>                           ::=:universal-preconditions (forall (<typed list(variable)>) <pre-GD>)
<pref-GD>                          ::=:preferences (preference [<pref-name>] <GD>)
<pref-GD>                          ::= <GD>
<pref-name>                        ::= <name>
<GD>                               ::= <atomic formula(term)>
<GD>                               ::=:negative-preconditions <literal(term)>
<GD>                               ::= (and <GD>*)
<GD>                               ::=:disjunctive-preconditions (or <GD>*)
<GD>                               ::=:disjunctive-preconditions (not <GD>)
<GD>                               ::=:disjunctive-preconditions (imply <GD> <GD>)
<GD>                               ::=:existential-preconditions (exists (<typed list(variable)>) <GD> )
<GD>                               ::=:universal-preconditions (forall (<typed list(variable)>) <GD> )
<GD>                               ::=:numeric-fluents <f-comp>
<f-comp>                           ::= (<binary-comp> <f-exp> <f-exp>)
<literal(t)>                       ::= <atomic formula(t)>
<literal(t)>                       ::= (not <atomic formula(t)>)
<atomic formula(t)>                ::= (<predicate> t*)
<atomic formula(t)>                ::=:equality (= t t)
<term>                             ::= <name>
<term>                             ::= <variable>
<term>                             ::=:object-fluents <function-term>
<function-term>                    ::=:object-fluents (<function-symbol> <term>*)
<f-exp>                            ::=:numeric-fluents <number>
<f-exp>                            ::=:numeric-fluents (<binary-op> <f-exp> <f-exp>)
<f-exp>                            ::=:numeric-fluents (- <f-exp>)
<f-exp>                            ::=:numeric-fluents <f-head>
<f-head>                           ::= (<function-symbol> <term>*)
```

Megjegyzés [ED1]: I put these 3 lines a little bit higher here compared to where they were in the original PDDL definition (because it may be more logical this way).

Megjegyzés [ED2]: This row is not necessary if `<primitive-type>` can be `object` (this would be the best, but since the earliest definitions of PDDL this was always left out somehow, and it is interesting not only because of version 3.1's object-fluents).

Megjegyzés [ED3]: This wasn't yet explicitly declared (in neither version of PDDL). So this is a correction.

Megjegyzés [ED4]: The definition of the built-in, 2-ary = predicate in case of the `:equality` requirement is given. …somehow this was also left out from PDDL definitions until now. So this is also a correction here.

Megjegyzés [DLK5]: There is no `<multi-op>` version here such as in the problem definition, among the rules for `<metric-f-exp>` (cf. 1.2).

```
<f-head>                         ::= <function-symbol>
<binary-op>                      ::= <multi-op>
<binary-op>                      ::= −
<binary-op>                      ::= /
<multi-op>                       ::= *
<multi-op>                       ::= +
<binary-comp>                    ::= >
<binary-comp>                    ::= <
<binary-comp>                    ::= =
<binary-comp>                    ::= >=
<binary-comp>                    ::= <=
<name>                           ::= Any string of characters.
<number>                         ::= Any numeric literal (integers and floats of form n.n).
<effect>                         ::= (and <c-effect>*)
<effect>                         ::= <c-effect>
<c-effect>                       ::=:conditional-effects (forall (<typed list (variable)>*) <effect>)
<c-effect>                       ::=:conditional-effects (when <GD> <cond-effect>)
<c-effect>                       ::= <p-effect>
<p-effect>                       ::= (<assign-op> <f-head> <f-exp>)
<p-effect>                       ::= (not <atomic formula(term)>)
<p-effect>                       ::= <atomic formula(term)>
<p-effect>                       ::=:numeric-fluents (<assign-op> <f-head> <f-exp>)
<p-effect>                       ::=:object-fluents (assign <function-term> <term>)
<p-effect>                       ::=:object-fluents (assign <function-term> undefined)
<cond-effect>                    ::= (and <p-effect>*)
<cond-effect>                    ::= <p-effect>
<assign-op>                      ::= assign
<assign-op>                      ::= scale-up
<assign-op>                      ::= scale-down
<assign-op>                      ::= increase
<assign-op>                      ::= decrease
<durative-action-def>            ::= (:durative-action <da-symbol>
                                          :parameters (<typed list (variable)>)
                                          <da-def body>)
<da-symbol>                      ::= <name>
<da-def body>                    ::= :duration <duration-constraint>
                                     :condition <emptyOr (da-GD)>
                                     :effect <emptyOr (da-effect)>
<da-GD>                          ::= <pref-timed-GD>
<da-GD>                          ::= (and <da-GD>*)
<da-GD>                          ::=:universal-preconditions (forall (<typed-list (variable)>) <da-GD>)
<pref-timed-GD>                  ::= <timed-GD>
<pref-timed-GD>                  ::=:preferences (preference [<pref-name>] <timed-GD>)
<timed-GD>                       ::= (at <time-specifier> <GD>)
<timed-GD>                       ::= (over <interval> <GD>)
<time-specifier>                 ::= start
<time-specifier>                 ::= end
<interval>                       ::= all
<duration-constraint>            ::=:duration-inequalities (and <simple-duration-constraint>+)
<duration-constraint>            ::= ()
<duration-constraint>            ::= <simple-duration-constraint>
<simple-duration-constraint>     ::= (<d-op> ?duration <d-value>)
<simple-duration-constraint>     ::= (at <time-specifier> <simple-duration-constraint>)
<d-op>                           ::=:duration-inequalities <=
<d-op>                           ::=:duration-inequalities >=
<d-op>                           ::= =
<d-value>                        ::= <number>
<d-value>                        ::=:numeric-fluents <f-exp>
<da-effect>                      ::= (and <da-effect>*)
<da-effect>                      ::= <timed-effect>
<da-effect>                      ::=:conditional-effects (forall (<typed list (variable)>) <da-effect>)
<da-effect>                      ::=:conditional-effects (when <da-GD> <timed-effect>)
<da-effect>                      ::=:numeric-fluents (<assign-op> <f-head> <f-exp-da>)
<timed-effect>                   ::= (at <time-specifier> <cond-effect>)
<timed-effect>                   ::=:numeric-fluents (at <time-specifier> <f-assign-da>)
<timed-effect>                   ::=:continuous-effects + :numeric-fluents (<assign-op-t> <f-head> <f-exp-t>)
<f-assign-da>                    ::= (<assign-op> <f-head> <f-exp-da>)
<f-exp-da>                       ::= (<binary-op> <f-exp-da> <f-exp-da>)
<f-exp-da>                       ::= (- <f-exp-da>)
<f-exp-da>                       ::=:duration-inequalities ?duration
<f-exp-da>                       ::= <f-exp>
```

Megjegyzés [DLK6]: This part is overly underspecified. For example may `<name>` be a PDDL domain description? Absolutely not. So a correction is needed here to specify exactly what we allow and what we do not allow for names and numbers. The correction can be based e.g. on Florent Teichteil-Königsbuch's paper from ICAPS-2008, titled "*Extending PPDDL1.0 to Model Hybrid Markov Decision Processes*". The need for such a correction was suggested by Éric Jacopin.

Megjegyzés [DLK7]: This must be an error here... Should be deleted (the *).

Megjegyzés [ED8]: 3 rows higher the same is given, but in an unconditional form… So I think that is not needed, and only this row here is necessary. This is so since the article describing the BNF of PDDL 2.1 (e.g. in the BNF of PDDL 3.0). Comment: „…numeric fluents, which … start being `undefined`… never become undefined again once a value has been assigned…"

Megjegyzés [ED9]: „Note that `undefined` is *not* a term, so it cannot be referred to in conditions."

Megjegyzés [DLK10]: This row stems from the BNF-specification of PDDL 2.1. The problem with it is that it would allow a durative action to have a numeric effect, which is **not temporally annotated** although the paper introducing PDDL 2.1 states that „*All conditions and effects of durative actions must be temporally annotated*" (see. Section 5, under Figure 6). This is an error here... So this line needs to be deleted. Two lines below it temporally annotated numeric effects are allowed.

Megjegyzés [DLK11]: Instead of this in the paper describing PDDL 2.1 and also 3.0 `<a-effect>` is written, which is not defined. The choice of `<cond-effect>` is eventually the result of discussions with *Gabriele Röger* and *Derek Long*. Otherwise `<p-effect>` or just `<effect>` could also be a candidate, but they would respectively either overly simplify or overly complicate the intended syntax, not to speak of the underlying semantics, which should be definite in any case. So this here is also a correction.

Megjegyzés [ED12]: This conditional requirement of this row wasn't part of neither specification, so it is now a correction...

Megjegyzés [ED13]: This was also left out from prev. specifications: correction.

Megjegyzés [DLK14]: Similarly to the non-durative case (see. comment DLK5), there is again no `<multi-op>` version here such as in the problem definition, among the rules for `<metric-f-exp>` (cf. 1.2).

```
<assign-op-t>                          ::= increase
<assign-op-t>                          ::= decrease
<f-exp-t>                              ::= (* <f-exp> #t)
<f-exp-t>                              ::= (* #t <f-exp>)
<f-exp-t>                              ::= #t
<derived-def>                          ::= (:derived <typed list (variable)> <GD>)
```

1.2 Problem description

```
<problem>                              ::= (define (problem <name>)
                                                 (:domain <name>)
                                                 [<require-def>]
                                                 [<object declaration>]
                                                 <init>
                                                 <goal>
                                                 [<constraints>]:constraints
                                                 [<metric-spec>]:numeric-fluents
                                                 [<length-spec>])
<object declaration>                   ::= (:objects <typed list (name)>)
<init>                                 ::= (:init <init-el>*)
<init-el>                              ::= <literal(name)>
<init-el>                              ::=:timed-initial-literals (at <number> <literal(name)>)
<init-el>                              ::=:numeric-fluents (= <f-head> <number>)
<init-el>                              ::=:object-fluents (= <basic-function-term> <name>)
<basic-function-term>                  ::= <function-symbol>
<basic-function-term>                  ::= (<function-symbol> <name>*)
<goal>                                 ::= (:goal <pre-GD>)
<constraints>                          ::=:constraints (:constraints <pref-con-GD>)
<pref-con-GD>                          ::= (and <pref-con-GD>*)
<pref-con-GD>                          ::=:universal-preconditions (forall (<typed list (variable)>) <pref-con-GD>)
<pref-con-GD>                          ::=:preferences (preference [<pref-name>] <con-GD>)
<pref-con-GD>                          ::= <con-GD>
<con-GD>                               ::= (and <con-GD>*)
<con-GD>                               ::= (forall (<typed list (variable)>) <con-GD>)
<con-GD>                               ::= (at end <GD>)
<con-GD>                               ::= (always <GD>)
<con-GD>                               ::= (sometime <GD>)
<con-GD>                               ::= (within <number> <GD>)
<con-GD>                               ::= (at-most-once <GD>)
<con-GD>                               ::= (sometime-after <GD> <GD>)
<con-GD>                               ::= (sometime-before <GD> <GD>)
<con-GD>                               ::= (always-within <number> <GD> <GD>)
<con-GD>                               ::= (hold-during <number> <number> <GD>)
<con-GD>                               ::= (hold-after <number> <GD>)
<metric-spec>                          ::=:numeric-fluents (:metric <optimization> <metric-f-exp>)
<optimization>                         ::= minimize
<optimization>                         ::= maximize
<metric-f-exp>                         ::= (<binary-op> <metric-f-exp> <metric-f-exp>)
<metric-f-exp>                         ::= (<multi-op> <metric-f-exp> <metric-f-exp>+)
<metric-f-exp>                         ::= (- <metric-f-exp>)
<metric-f-exp>                         ::= <number>
<metric-f-exp>                         ::= (<function-symbol> <name>*)
<metric-f-exp>                         ::= <function-symbol>
<metric-f-exp>                         ::= total-time
<metric-f-exp>                         ::=:preferences (is-violated <pref-name>)
<length-spec>                          ::= (:length [(:serial <integer>)] [(:parallel <integer>)])
                                           The length-spec is deprecated since PDDL 2.1.
```

1.2.1 Lifting restrictions (from constraint declaration)

```
<con-GD>                               ::= (always <con-GD>)
<con-GD>                               ::= (sometime <con-GD>)
<con-GD>                               ::= (within <number> <con-GD>)
<con-GD>                               ::= (at-most-once <con-GD>)
<con-GD>                               ::= (sometime-after <con-GD> <con-GD>)
<con-GD>                               ::= (sometime-before <con-GD> <con-GD>)
<con-GD>                               ::= (always-within <number> <con-GD> <con-GD>)
<con-GD>                               ::= (hold-during <number> <number> <con-GD>)
<con-GD>                               ::= (hold-after <number> <con-GD>)
```

Megjegyzés [ED15]: These rules are part of the BNF definition describing PDDL 2.1, but were somehow left out from the definition of PDDL 3.0…

Megjegyzés [DLK16]: This is an error, since there is no mention of the related `<predicate>`. This is so since PDDL2.2 in [3] (see. Sec. 2.1 and A.4) and PDDL 3.0 in [4] (see. Sec. 2.4). [3] would suggest maybe `<atomic formula(term)>` instead of `<typed list(variable)>`, but then there would be no types in the head of the derived-rule. To include both the name of the related predicate and the types of variables we'd need to write `<atomic formula skeleton>`. Thanks for bringing this issue to my attention to Ron Alford.

Megjegyzés [ED17]: This row allows us to add negated facts to the initial state.

Megjegyzés [DLK18]: PDDL 2.2 defined this so that it can have only facts, and not for example value assignments to numerical, or now even object fluents...

Megjegyzés [DLK19]: This may not be grounded, thus this is an error. It is part of the language since PDDL 2.1. For simplicity `<basic-function-term>` may be used instead.

Megjegyzés [ED20]: This here is in a bit different order than on the IPC6 webpage http://ipc.informatik.uni-freiburg.de/PddlExtension (at the „*Initial values for object fluents*" part). Comment.: `<basic-function-term>` should be part of the problem description and not the domain description (since the domain description is never referring to it in contrary to the problem description).

Megjegyzés [ED21]: There was a typing error in the PDDL 3.0 definition („::" instead of „::="). Correction.

Megjegyzés [ED22]: If we wish to embed modal operators into each other, then instead of these rules we should use those in section 1.2.1.

Megjegyzés [ED23]: This was left out from the PDDL articles… Correction.

Megjegyzés [ED24]: Before the PDDL 3.0 article this was `<ground-f-exp>`.

Megjegyzés [DLK25]: It is important to observe, that in contrary to `f-exp`, here is `name`, and not `term` (for the `metric` to be grounded, without variables, objects...).

Megjegyzés [ED26]: This was left out from the PDDL 3.0 article... Correction.

Megjegyzés [ED27]: This also was left out from the PDDL 3.0 (and from PDDL 2.2 too), but it was part of the PDDL 2.1.

Megjegyzés [DLK28]: This is ill defined, since if these rules would be put in the grammar, then there would be no normal end to the recursive embedding of the modal operators. `con-GD` should be somehow transformed to `GD` too, but not directly, since then it would be equal (`con-GD` should not be equal to `GD`).

1.3 Requirements

Here is a table of all requirements in PDDL3.1. Some requirements imply others; some are abbreviations for common sets of requirements. If a domain stipulates no requirements, it is assumed to declare a requirement for `:strips`.

`:strips`	Basic STRIPS-style adds and deletes
`:typing`	Allow type names in declarations of variables
`:negative-preconditions`	Allow `not` in goal descriptions
`:disjunctive-preconditions`	Allow `or` in goal descriptions
`:equality`	Support `=` as built-in predicate
`:existential-preconditions`	Allow `exists` in goal descriptions
`:universal-preconditions`	Allow `forall` in goal descriptions
`:quantified-preconditions`	= `:existential-preconditions` + `:universal-preconditions`
`:conditional-effects`	Allow `when` in action effects
`:fluents`	= `:numeric-fluents` + `:object-fluents`
`:numeric-fluents`	Allow numeric function definitions and use of effects using assignment operators and arithmetic preconditions.
`:adl`	= `:strips` + `:typing` + `:negative-preconditions` + `:disjunctive-preconditions` + `:equality` + `:quantified-preconditions` + `:conditional-effects`
`:durative-actions`	Allows durative actions. Note that this does not imply `:numeric-fluents`.
`:duration-inequalities`	Allows duration constraints in durative actions using inequalities.
`:continuous-effects`	Allows durative actions to affect fluents continuously over the duration of the actions.
`:derived-predicates`	Allows predicates whose truth value is defined by a formula
`:timed-initial-literals`	Allows the initial state to specify literals that will become true at a specified time point. Implies `:durative-actions`
`:preferences`	Allows use of preferences in action preconditions and goals.
`:constraints`	Allows use of constraints fields in domain and problem files. These may contain modal operators supporting trajectory constraints.
`:action-costs`	If this requirement is included in a PDDL specification, the use of numeric fluents is enabled (similar to the `:numeric-fluents` requirement). However, numeric fluents may only be used in certain very limited ways: 1. Numeric fluents may not be used in any conditions (preconditions, goal conditions, conditions of conditional effects, etc.). 2. A numeric fluent may only be used as the target of an effect if it is 0-ary and called `total-cost`. If such an effect is used, then the `total-cost` fluent must be explicitly initialized to 0 in the initial state. 3. The only allowable use of numeric fluents in effects is in effects of the form `(increase (total-cost) <numeric-term>)`, where the `<numeric-term>` is either a non-negative numeric constant or of the form `(<function-symbol> <term>*)`. (The `<term>` here is interpreted as shown in the PDDL grammar, i.e. it is a variable symbol or an object constant. Note that this `<term>` cannot be a `<function-term>`, even if the object fluents requirement is used.) 4. No numeric fluent may be initialized to a negative value. 5. If the problem contains a `:metric` specification, the objective must be `(minimize (total-cost))`, or - only if the `:durative-actions` requirement is also set - to minimize a linear combination of `total-cost` and `total-time`, with non-negative coefficients. Note that an action can have multiple effects that increase `(total-cost)`, which is particularly useful in the context of conditional effects. Also note that these restrictions imply that `(total-cost)` never decreases throughout plan execution, i.e., action costs are never negative.

Megjegyzés [ED29]: This was also left out from the PDDL 3.0 article (and from PDDL 2.2 too), but it is part of PDDL 2.1. Comment: are `:fluents` or `:numeric-fluents` requirements implied by these two highlighted requirements? - it would be logical.

References

[1] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M.; Weld, D., Wilkins, D. (1998). ***PDDL**---The Planning Domain Definition Language*. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

[2] Fox M., Long D. (2003). ***PDDL2.1**: An Extension to pddl for Expressing Temporal Planning Domains*, Journal of Artificial Intelligence Research 20: 61-124.

[3] Edelkamp S., Hoffmann J. (2004). ***PDDL2.2**: The Language for the Classical Part of the 4th International planning Competition*, Technical Report No. 195, Institut für Informatik.

[4] Gerevini, A. Long D. (2005). *BNF Description of **PDDL3.0***. Unpublished manuscript from the IPC-5 website.

[5] Helmert, M. (2008). *Changes in **PDDL 3.1***. http://ipc.informatik.uni-freiburg.de/PddlExtension