bni: A PDDL to C compiler with integrated REPL for interactive testing

BRACIS — ENIAC

Bruno Campos Ribeiro & Igor e Silva Penha
Supervisor: Bruno Cesar Ribas

University of Brasília

September 29, 2025

UnB

SΛT
Planning Group

Summary

Introduction

Parser

REPL

Validator

References

Introduction

Motivation I

- **Automated Planning** is a essential branch in AI, robotics, and complex systems;
- **PDDL** is the standard language for domain and problem modelling;

Motivation II

- It arose from the curiosity to explore the feasibility of compiling a PDDL model into a C programme. This approach could open up new possibilities for an interactive and compiled planner in C, integrating domain-dependent and domain-independent planners;
- Need for an **efficient, portable, and integrated** solution for parsing, testing, and validation.

Related Work

- VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL [Howey, Long e Fox 2004].
- SymbolicPlanners.jl library (Julia Planner) [Zhi-Xuan 2022].

Goals

- Implement a modular parser in C for PDDL domains and problems;
- Create an interactive REPL interface for incremental testing and debugging;
- Incorporate incremental and *post-hoc* validation of plans;

Modular Parser

- Translation of PDDL domains and problems into data structures in C;
- Efficient and manipulable representation of actions, states, and goals;
- Main benefits:
  - Transparency;
  - Performance;
  - Portability.

Constants & Objects

Listing 1: PDDL objects

```
(:objects
    f—stop s—stop t—stop — stop)
(:constants
    ICE—CREAM—PARLOUR — stop)
```

Listing 2: Objects representation in .h file

```
enum t_stop {
  f_stop,
  s_stop,
  t_stop,
  LENGTH_stop
};
typedef struct stopMap {
  const char *str;
  enum t_stop value;
}stopMap;
```

Listing 3: Objects in .c file

```
stopMap stop_map[LENGTH_stop] = {
  {"f_stop", f_stop},
  {"s_stop", s_stop},
  {"t_stop", t_stop},
};
const char*
get_stop_names(enum t_stop e) {
  if (e >= 0 && e < LENGTH_stop)
    return stop_map[e].str;
  return NULL;
}
enum
t_stop get_stop_enum(const char *s)
{
  if (s == NULL)
    return LENGTH_stop;
  for (int i=0;i!=LENGTH_stop;i++)
    if (strcmp(s,stop_map[i].str)
         ==0)
      return stop_map[i].value;
  return LENGTH_stop;
}
```

Predicates

Listing 4: PDDL predicates

```
(:predicates
  (connected ?s1 ?s2 - stop)
  (i-am-at ?s - stop)
  (order-ice-cream ?i - ice-cream)
  (has-ice-cream ?i - ice-cream)
  (stop-is ?s1 ?s2 - stop))
```

Listing 5: Predicates in C

```
bool connected[4][4];
bool i_am_at[4];
bool passed_through[4];
bool order_ice_cream[7];
bool has_ice_cream[7];
bool stop_is[4][4];
bool checktrue_connected(int s1, int s2) {
  return connected[s1][s2];
}
bool checktrue_i_am_at(int s) {
  return i_am_at[s];
}
bool checktrue_order_ice_cream(int i) {
  return order_ice_cream[i];
}
bool checktrue_has_ice_cream(int i) {
  return has_ice_cream[i];
}
bool checktrue_stop_is(int s1, int s2) {
  return stop_is[s1][s2];
}
```

Actions — Parameters

Listing 6: PDDL parameters

```
(:action TRAVEL
  :parameters (?s1 ?s2 - stop)
(:action BUY-ICE-CREAM
  :parameters
    (?s - stop ?i - ice-cream)
```

Listing 7: Parameters in C

```
struct travel {
  enum stop s2;
  enum stop s1;
};
struct buy_ice_cream {
  enum stop s;
  enum ice_cream i;
};
```

Actions — Precondition

Listing 8: PDDL preconditions

```
(:action TRAVEL
  :precondition (and
    (i-am-at ?s1)
    (or (connected ?s1 ?s2)
        (connected ?s2 ?s1)))
(:action BUY-ICE-CREAM
  :precondition (and
    (stop-is ?s ICE-CREAM-PARLOUR)
    (has-ice-cream ?i))
```

Listing 9: Precondition in C

```
bool
checktrue_travel(struct travel s) {
  return (checktrue_i_am_at(s.s1)
       and
  (checktrue_connected(s.s1,s.s2)
       or
  checktrue_connected(s.s2,s.s1)));
}

bool
checktrue_buy_ice_cream (struct
     buy_ice_cream s) {
  return (checktrue_stop_is(s.s,
       ice_cream_parlour) and
    checktrue_has_ice_cream(s.i));
}
```

Actions — Effect

Listing 10: PDDL effect

```
(:action TRAVEL
  :effect (and
    (not (stop-is ?s1 ?s1))
    (not (i-am-at ?s1))
    (stop-is ?s2 ?s2)
    (i-am-at ?s2))
(:action BUY-ICE-CREAM
  :effect (and
    (order-ice-cream ?i)
    (not (has-ice-cream ?i)))
```

Listing 11: Effect in C

```
void
apply_travel(struct travel s) {
  i_am_at[s.s2] = 1;
  i_am_at[s.s1] = 0;
  stop_is[s.s1][s.s1] = 0;
  stop_is[s.s2][s.s2] = 1;
}
void
apply_buy_ice_cream(struct
     buy_ice_cream s) {
  order_ice_cream[s.i] = 1;
  has_ice_cream[s.i] = 0;
}
```

Actions — Init

Listing 12: PDDL init

```
(:init (connected f-stop s-stop)
       (connected s-stop t-stop)
       (connected t-stop
            ICE-CREAM-PARLOUR)
       (i-am-at f-stop)
       (has-ice-cream vanilla)
       (has-ice-cream chocolate)
       (has-ice-cream strawberry))
```

Listing 13: Init in C

```
void initialize(void) {
  connected[f_stop][s_stop] = true;
  connected[s_stop][t_stop] = true;
  connected[t_stop][
       ice_cream_parlour] = true;
  i_am_at[f_stop] = true;
  has_ice_cream[vanilla] = true;
  has_ice_cream[chocolate] = true;
  has_ice_cream[strawberry] = true;
}
```

Actions — Goal

Listing 14: PDDL goal

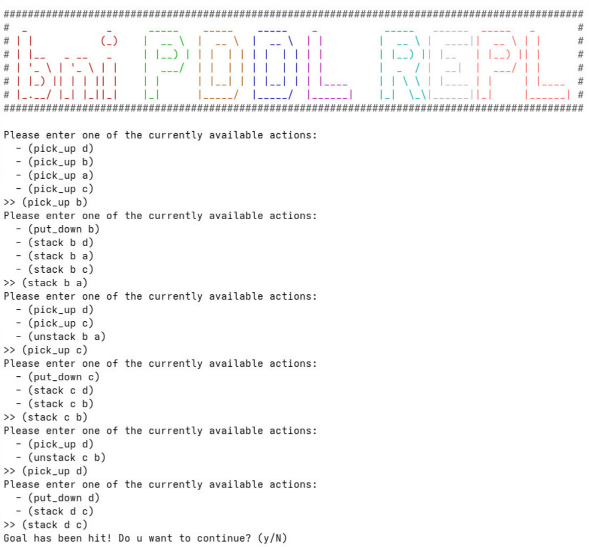
```
(:goal (forall (?i - ice-cream)
   (not (has-ice-cream ?i))))
```

Listing 15: Goal in C

```
bool checktrue_goal(void) {
  bool forall1 = true;
  for (int i0 = 0; i0 <
       LENGTH_ice_cream; i0++) {
    if(checktrue_has_ice_cream(i0))
    { forall1 = false; break; }
  }
  return (forall1);
}
```

Interactive REPL

- Incremental testing and debugging of PDDL domains;
- Real-time validation of actions and states;
- Support for interactive exploration of states and plans.



```
################################################################################################
#  _                 _    _____     _____     _____     _           _____   ______  _____   _       #
# | |               (_)  |  __ \   |  __ \   |  __ \   | |         |  __ \ |  ____||  __ \ | |       #
# | |__   _ __   _       | |__) |  | |  | |  | |  | |  | |         | |__) || |__   | |__) || |       #
# | '_ \ | '_ \ | |      |  ___/   | |  | |  | |  | |  | |         |  _  / |  __|  |  ___/ | |       #
# | |_) || | | || |      | |       | |__| |  | |__| |  | |____     | | \ \ | |____ | |     | |____   #
# |_.__/ |_| |_||_|      |_|       |_____/   |_____/   |______|    |_|  \_\|______||_|     |______| #
################################################################################################

Please enter one of the currently available actions:
  - (pick_up d)
  - (pick_up b)
  - (pick_up a)
  - (pick_up c)
>> (pick_up b)
Please enter one of the currently available actions:
  - (put_down b)
  - (stack b d)
  - (stack b a)
  - (stack b c)
>> (stack b a)
Please enter one of the currently available actions:
  - (pick_up d)
  - (pick_up c)
  - (unstack b a)
>> (pick_up c)
Please enter one of the currently available actions:
  - (put_down c)
  - (stack c d)
  - (stack c b)
>> (stack c b)
Please enter one of the currently available actions:
  - (pick_up d)
  - (unstack c b)
>> (pick_up d)
Please enter one of the currently available actions:
  - (put_down d)
  - (stack d c)
>> (stack d c)
Goal has been hit! Do u want to continue? (y/N)
```

Figure: Blocks World REPL Simulation

Validator

Plan Validation

- The *bni* system incorporates built-in support for plan validation, which verifies a given sequence of actions correctly transitions the system from the initial state to the goal state while respecting all specified preconditions and effects.

```
$ ./bni.sh --validate plan domain.pddl[1] problem.pddl[2]
```

[1] Domain source code

[2] Problem source code

Valid Plan

Listing 16: Blocks instance-1 Valid Plan

```
0 : (pick-up b)
1 : (stack b a)
2 : (pick-up c)
3 : (stack c b)
4 : (pick-up d)
5 : (stack d c)
```

```
$ ./bni.sh --validate plan domain.pddl problem.pddl
VALID PLAN
$
```

Incompet Plan

Listing 17: Blocks instance-1 Incomplet Plan

```
0 : (pick-up b)
1 : (stack b a)
2 : (pick-up c)
3 : (stack c b)
4 : (pick-up d)
;5 : (stack d c) comment the last move
```

```
$ ./bni.sh --validate plan domain.pddl problem.pddl
INCOMPLET PLAN
$
```

Spelling Error Plan

Listing 18: Blocks instance-1 with Spelling Error

```
0 : (pik-up b) ; <- remove letter 'c' from pick
1 : (stack b a)
2 : (pick-up c)
3 : (stack c b)
4 : (pick-up d)
5 : (stack d c)
```

```
$ ./bni.sh --validate plan domain.pddl problem.pddl
ATTENTION: Unrecognised command.  Check spelling and
try again.
$
```

Invalid Plan

Listing 19: Blocks instance-1 Invalid Plan

```
0 : (pick-up b)
1 : (stack b a)
2 : (pick-up c)
3 : (stack c b)
;4 : (pick-up d) invalid to make the last move
5 : (stack d c)
```

```
$ ./bni.sh --validate plan domain.pddl problem.pddl
ATTENTION: Action with invalid parameters.
$
```

References I

HOWEY, R.; LONG, D.; FOX, M. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In: *16th IEEE International Conference on Tools with Artificial Intelligence*. [S.l.: s.n.], 2004. p. 294–301.

ZHI-XUAN, T. MS Thesis, *PDDL.jl: An Extensible Interpreter and Compiler Interface for Fast and Flexible AI Planning*. 2022.

Thank You!

Bruno Ribeiro | Igor Penha | Bruno Ribas