Finding Parallel Regions with Temporal Planning

Claudio Scheer[1]

[1]Pontifical Catholic University of Rio Grande do Sul - PUCRS
claudio.scheer@edu.pucrs.br

Final Presentation, July 2020

Table of Contents

Table of Contents

Parallel Regions

- Find parallel regions in a source code;

Common Approach

Static analysis of the source code:

- loops;
- instruction dependencies;
- identifying whether the arguments are read or written;

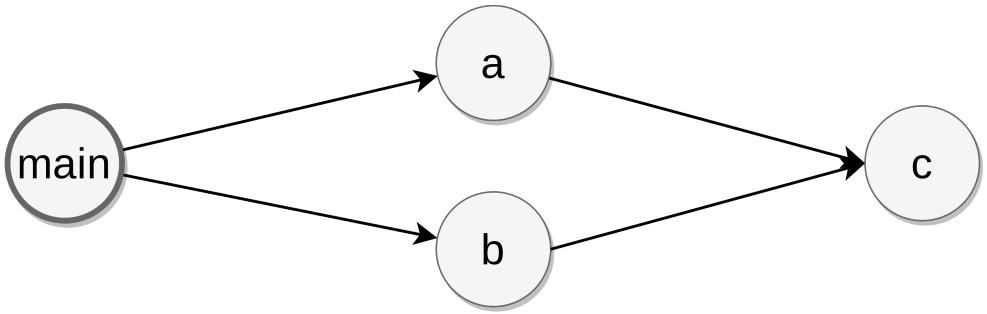
Table of Contents

Approach

- PDDL domain file executes the instructions;
- PDDL problem file defines the instruction dependency tree;
- Simultaneous Temporal Planner finds a temporal plan;
  - State that minimizes the total cost;

PDDL Problem - 1

```
int main()
{
  int a = 3;
  int b = 3;
  int c = a + b;
  return 0;
}
```



PDDL Problem - 1

```
(:init
  (executed_instruction id0)

  (assignment_id assignmentA id1)
  (assignment_id assignmentB id2)
  (operation_id sumAB id3)
  (assignment_id assignmentC id4)

  (dependency_tree id0 id1)
  (dependency_tree id0 id2)
  (dependency_tree id1 id3)
  (dependency_tree id2 id3)
  (dependency_tree id3 id4)
)

(:goal (and
  (executed_assignment assignmentA)
  (executed_assignment assignmentB)
  (executed_binary_operation assignmentA assignmentB sumAB
      assignmentC)
  (executed_assignment assignmentC)
))
```

PDDL Domain - assignment

```
(:durative-action assignment
  :parameters (?instruction_id - id ?id - assignment)
  :duration (= ?duration 1)
  :condition (and
    (at start (assignment_id ?id ?instruction_id))
    (at start (not (executed_assignment ?id)))
    (at start (forall (?parent - id)
      (or
        (not (dependency_tree ?parent ?instruction_id))
        (executed_instruction ?parent)
      )
    ))
  )
  :effect (and
    (at end (executed_instruction ?instruction_id))
    (at end (executed_assignment ?id))
  )
)
```

PDDL Domain - binary_operation

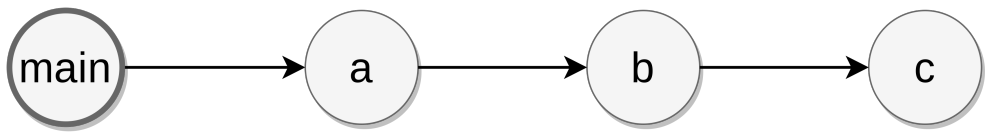
```
(:durative-action binary_operation
  :parameters (
    ?instruction_id - id ?idA - assignment
    ?idB - assignment ?operation_id - operation ?idC - assignment
  )
  :duration (= ?duration 1)
  :condition (and
    (at start (operation_id ?operation_id ?instruction_id))
    (at start (forall (?parent - id)
      (or
        (not (dependency_tree ?parent ?instruction_id))
        (executed_instruction ?parent)
      )
    ))
    (at start (not (executed_operation ?operation_id)))
    (at start (not (executed_binary_operation ?idA ?idB ?operation_id ?idC)))
    (at start (executed_assignment ?idA))
    (at start (executed_assignment ?idB))
  )
  :effect (and
    (at end (executed_instruction ?instruction_id))
    (at end (executed_operation ?operation_id))
    (at end (executed_binary_operation ?idA ?idB ?operation_id ?idC))
  )
)
```

PDDL Problem - 1

0.000	1.000	2.000
assignmentA		
assignmentB		
	sumAB	
		assignmentC

PDDL Problem - 2

```
int main()
{
  int a = 3;
  int b = a + 1;
  int c = a + b;
  return 0;
}
```



PDDL Problem - 2

```
(:init
  (executed_instruction id0)

  (assignment_id assignmentA id1)
  (assignment_id assignmentB id2)
  (operation_id sumAB id3)
  (assignment_id assignmentC id4)

  (dependency_tree id0 id1)
  (dependency_tree id1 id2)
  (dependency_tree id1 id3)
  (dependency_tree id2 id3)
  (dependency_tree id3 id4)
)

(:goal (and
  (executed_assignment assignmentA)
  (executed_assignment assignmentB)
  (executed_binary_operation assignmentA assignmentB sumAB
      assignmentC)
  (executed_assignment assignmentC)
))
```

PDDL Problem - 2

0.000	1.000	2.000	3.000
assignmentA			
	assignmentB		
		sumAB	
			assignmentC

Table of Contents

How to handle `for` loops?

```
int main()
{
  int s = 0;
  std::vector<int> x = {1, 2, 3};
  for (int i = 0; i < x.size(); i++)
  {
    s += x[i];
  }
  return 0;
}
```

```
int main()
{
  int a[3] = {0};
  a[0] = rand();
  for (int i = 1; i < 3; ++i)
  {
    a[i] = a[i - 1] + rand();
  }
  return 0;
}
```

PDDL Problem

```
(:init
  (executed_instruction id0)

  (assignment_id assignmentS id1)
  (assignment_id assignmentArray id2)
  (operation_id sum0 id3)
  (assignment_id assignmentS0 id4)
  (operation_id sum1 id5)
  (assignment_id assignmentS1 id6)
  (operation_id sum2 id7)
  (assignment_id assignmentS2 id8)

  (dependency_tree id0 id1)
  (dependency_tree id0 id2)
  (dependency_tree id1 id3)
  (dependency_tree id2 id3)
  (dependency_tree id3 id4)
  (dependency_tree id1 id5)
  (dependency_tree id2 id5)
  (dependency_tree id5 id6)
  (dependency_tree id1 id7)
  (dependency_tree id2 id7)
  (dependency_tree id7 id8)
)
```

```
(:goal (and
  (executed_assignment assignmentS)
  (executed_assignment assignmentArray)
  (executed_binary_operation assignmentS
        assignmentArray sum0 assignmentS0)
  (executed_assignment assignmentS0)
  (executed_binary_operation assignmentS
        assignmentArray sum1 assignmentS1)
  (executed_assignment assignmentS1)
  (executed_binary_operation assignmentS
        assignmentArray sum2 assignmentS2)
  (executed_assignment assignmentS2)
))
```

stp-2

0.000: (assignment id2 assignmentarray)
0.000: (assignment id1 assignments)
1.002: (binary_operation id3 assignments assignmentarray sum0 assignments0)
1.002: (binary_operation id5 assignments assignmentarray sum1 assignments1)
2.002: (binary_operation id7 assignments assignmentarray sum2 assignments2)
3.002: (assignment id4 assignments0)
3.002: (assignment id6 assignments1)
4.002: (assignment id8 assignments2)

stp-3

0.000: (assignment id2 assignmentarray)
0.000: (assignment id1 assignments)
1.002: (binary_operation id3 assignments assignmentarray sum0 assignments0)
1.002: (binary_operation id5 assignments assignmentarray sum1 assignments1)
1.002: (binary_operation id7 assignments assignmentarray sum2 assignments2)
2.002: (assignment id4 assignments0)
2.002: (assignment id6 assignments1)
3.002: (assignment id8 assignments2)

Table of Contents

Final Remarks

- It is possible to identify parallel instructions;

- We need to inform the dependency tree;
- How to parse the source code to a PDDL problem?