Modeling and Proving the Parallel Climbers Puzzle in Event-B

Kevin Schwarz

Heinrich Heine University Düsseldorf, Universitätsstr. 1, 40225 Düsseldorf, Germany
kevin.schwarz@hhu.de

1 Modeling

The Parallel Climbers Puzzle as described in [1] is modeled as follows: It consists of a total function `V` representing the mountain range and two climbers, left and right, which traverse the mountain range, maintaining the same height at any given moment. Start and end point of the mountain range are located at height 0, all other points must be higher. There is a point `m`, which is the highest point on the mountain range, all other points must be lower. The two climbers will always meet at that specific point `m`. The position of a climber is stored as an ordered pair of points in order to represent a vertex position or an edge position. There is a total function `Positions`, which maps the possible positions to height values according to `V`. Vertex positions are mapped to the vertex height and edge positions are mapped to an open interval between the two incident vertices. In addition, a relation `Valid_Positions` is defined to collate pairs of positions, which satisfy the two following conditions: The left climber must be left of point `m`, vice versa the right climber must be right of point `m` and the left and right climber must be at the same height (see Fig. 1 and Fig. 2).

In order to move the two climbers, there is a convergent event `move`, that determines all possible moves by checking if the next position pair is an element of `Valid_Positions`. If a climber is on a vertex position it can move to an incident edge position or an adjacent vertex position and if a climber is on an edge position it can only move to an incident vertex position. Furthermore, to prevent idleness or backtracking, only moves with progress towards point `m` are allowed (see Fig. 3). A Variant is provided to prove that the move event only makes progress (see Fig. 4).

2 Proving

The goal is to show that the Parallel Climbers Puzzle always has a solution for any given function `V`. In order to prove this, we must show that the convergent event `move` is always executable unless the desired target state is reached (see Fig. 5).

Proving it unfolds in many case distinctions, which must be proven individually because the auto prover and external tools like ProB Disprover [2] or SMT

Solver [3] cannot automatically prove the invariants. Nonetheless, these tools were very helpful in the broken down case distinctions.

Proving the invariants took extremely well-planned steps because of the sheer dimension of the proof. Having planned the proof and constructing it inside the interactive prover from Rodin took approximately 1 hour, as well for the other two invariants. Many steps inside the case distinctions were very similar but could not be made in advance. Doing this for approximately 16-20 case distinctions for every invariant, adds up to a lot of time spent just clicking through the planned steps.

However, there are cases that have not been proved by now. These cases are states, that are not reachable by the allowed moves (see Fig. 7). Proving them would possibly come down to a proof by contradiction.

```
n1: n ∈ N1 not theorem ›
v1: V ∈ 1‥n ⟶ N not theorem ›
v2: V(1) = 0 ∧ V(n) = 0 not theorem ›
v3: ∀i·i ∈ dom(V) ∧ i ≠ 1 ∧ i ≠ n ⇒ V(i) > 0 not theorem ›
v4: ∀i·i ∈ dom(V) ∧ i + 1 ∈ dom(V) ⇒ V(i) ≠ V(i + 1) not theorem ›
m1: m ∈ dom(V) not theorem ›
m2: ∀i·i ∈ dom(V) ∧ i ≠ m ⇒ V(m) > V(i) not theorem ›
pos1:   Positions ∈ {i↦j | i ∈ dom(V) ∧ j ∈ dom(V) ∧ (j = i ∨ j = i + 1)} ⟶ ℙ(N) not theorem ›
pos2:   ∀i·i ∈ dom(Positions)
          ⇒   (V(prj1(i)) < V(prj2(i)) ⇒ Positions(i) = {j | V(prj1(i)) < j ∧ j < V(prj2(i))})
            ∧ (V(prj1(i)) > V(prj2(i)) ⇒ Positions(i) = {j | V(prj2(i)) < j ∧ j < V(prj1(i))})
            ∧ (V(prj1(i)) = V(prj2(i)) ⇒ Positions(i) = {V(prj1(i))}) not theorem ›
valid1: Valid_Positions ∈ dom(Positions) ↔ dom(Positions) not theorem ›
valid2: Valid_Positions = {l↦r |
           l ∈ dom(Positions) ∧ r ∈ dom(Positions)
         ∧ prj2(l) ≤ m ∧ m ≤ prj1(r)
         ∧   (prj1(l) = prj2(l) ∧ prj1(r) = prj2(r)
                ⇒ V(prj1(l)) = V(prj1(r)))
           ∧ (prj1(l) = prj2(l) ∧ prj1(r) ≠ prj2(r)
                ⇒ V(prj1(l)) ∈ Positions(r))
           ∧ (prj1(l) ≠ prj2(l) ∧ prj1(r) = prj2(r)
                ⇒ V(prj1(r)) ∈ Positions(l))} not theorem ›
```

Fig. 1. Event-B context component

```
inv1:   left_climber↦right_climber ∈ Valid_Positions not theorem ›

INITIALISATION:    not extended ordinary ›
THEN
◦   act1:   left_climber ≔ 1↦1 ›
◦   act2:   right_climber ≔ n↦n ›
END
```

Fig. 2. Event-B machine component - Init

```
move:   not extended convergent ›
ANY
◦   next     ›
WHERE
◦   grd1:   next ∈ Valid_Positions not theorem ›
◦   grd2:   prj1(left_climber) = prj2(left_climber) ∧ prj1(right_climber) = prj2(right_climber)
              ⇒   next = prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber)    )
                  ∨ next = prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1)
                  ∨ next = prj1(left_climber)     ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1)
                  ∨ next = prj1(left_climber) − 1 ↦ prj2(left_climber)     ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1)
                  ∨ next = prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber)     ↦ prj2(right_climber) + 1)
             not theorem ›
◦   grd3:   prj1(left_climber) = prj2(left_climber) ∧ prj1(right_climber) ≠ prj2(right_climber)
              ⇒   next = prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber)     ↦ prj2(right_climber)    )
                  ∨ next = prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber)     ↦ prj2(right_climber) − 1)
                  ∨ next = prj1(left_climber)     ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber)     ↦ prj2(right_climber) − 1)
                  ∨ next = prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber) + 1 ↦ prj2(right_climber)    )
             not theorem ›
◦   grd4:   prj1(left_climber) ≠ prj2(left_climber) ∧ prj1(right_climber) = prj2(right_climber)
              ⇒   next = prj1(left_climber)     ↦ prj2(left_climber)     ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1)
                  ∨ next = prj1(left_climber) + 1 ↦ prj2(left_climber)     ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1)
                  ∨ next = prj1(left_climber) + 1 ↦ prj2(left_climber)     ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber)    )
                  ∨ next = prj1(left_climber)     ↦ prj2(left_climber) − 1 ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1)
             not theorem ›
THEN
◦   act1:   left_climber ≔ prj1(next) ›
◦   act2:   right_climber ≔ prj2(next) ›
END
```

Fig. 3. Event-B machine component - move event

```
vrn1:    prj1(right_climber) − prj1(left_climber) + prj2(right_climber) − prj2(left_climber) ›
```

Fig. 4. Event-B machine component - Variant

```
inv2:   prj1(left_climber) = prj2(left_climber) ∨ prj1(right_climber) = prj2(right_climber) not theorem ›
inv10:  left_climber ≠ right_climber
          ⇒   (prj1(left_climber) = prj2(left_climber) ∧ prj1(right_climber) = prj2(right_climber)
                 ⇒   prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber)    ) ∈ Valid_Positions
                     ∨ prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1) ∈ Valid_Positions
                     ∨ prj1(left_climber)     ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1) ∈ Valid_Positions
                     ∨ prj1(left_climber) − 1 ↦ prj2(left_climber)     ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1) ∈ Valid_Positions
                     ∨ prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber)     ↦ prj2(right_climber) + 1) ∈ Valid_Positions
             ) not theorem ›
inv11:  left_climber ≠ right_climber
          ⇒   (prj1(left_climber) = prj2(left_climber) ∧ prj1(right_climber) ≠ prj2(right_climber)
                 ⇒   prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber)     ↦ prj2(right_climber)    ) ∈ Valid_Positions
                     ∨ prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber)     ↦ prj2(right_climber) − 1) ∈ Valid_Positions
                     ∨ prj1(left_climber)     ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber)     ↦ prj2(right_climber) − 1) ∈ Valid_Positions
                     ∨ prj1(left_climber) + 1 ↦ prj2(left_climber) + 1 ↦ (prj1(right_climber) + 1 ↦ prj2(right_climber)    ) ∈ Valid_Positions
             ) not theorem ›
inv12:  left_climber ≠ right_climber
          ⇒   (prj1(left_climber) ≠ prj2(left_climber) ∧ prj1(right_climber) = prj2(right_climber)
                 ⇒   prj1(left_climber)     ↦ prj2(left_climber)     ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1) ∈ Valid_Positions
                     ∨ prj1(left_climber) + 1 ↦ prj2(left_climber)     ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1) ∈ Valid_Positions
                     ∨ prj1(left_climber) + 1 ↦ prj2(left_climber)     ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber)    ) ∈ Valid_Positions
                     ∨ prj1(left_climber)     ↦ prj2(left_climber) − 1 ↦ (prj1(right_climber) − 1 ↦ prj2(right_climber) − 1) ∈ Valid_Positions
             ) not theorem ›
```

Fig. 5. Event-B machine component - Invariants

Fig. 6. Proven POs

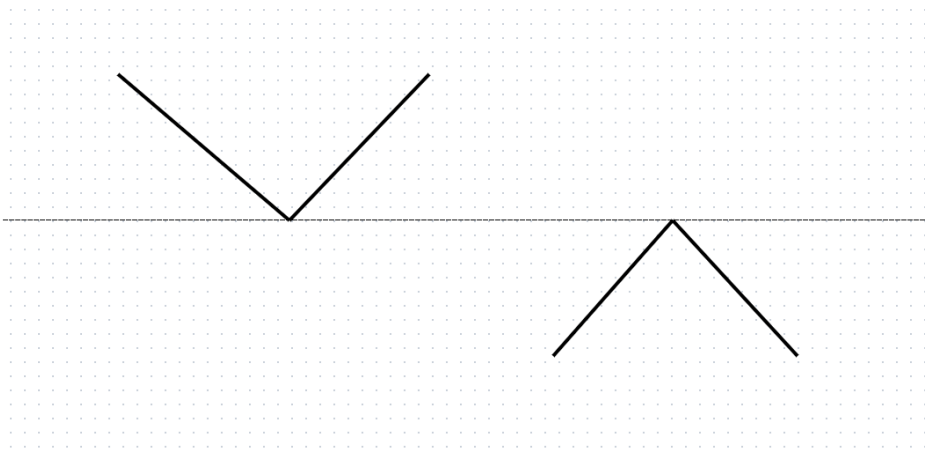

Fig. 7. Case that cannot be reached

References

1. Tucker, A. (1995). The Parallel Climbers Puzzle: A Case Study in the Power of Graph Models. Math Horizons, 3(2), 22–24. https://doi.org/10.1080/10724117.1995.11974954
2. Leuschel, M., Butler, M. (2003). ProB: A Model Checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds) FME 2003: Formal Methods. FME 2003. Lecture Notes in Computer Science, vol 2805. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-45236-2_46
3. Déharbe, D., Fontaine, P., Guyot, Y., Voisin, L. (2012). SMT Solvers for Rodin. In: Derrick, J., et al. Abstract State Machines, Alloy, B, VDM, and Z. ABZ 2012. Lecture Notes in Computer Science, vol 7316. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-30885-7_14