GDTs and proofs for Robots on Mars

Bruno Mermet Arnaud Saval Gaële Simon
Bruno Zanuttini *

October 19, 2006

Abstract

This document details the application of the GDT model to the scenario entitled "Robots on Mars". Full GDTs and proofs are given for both robots.

1 Introduction and notation

This paper describes in full details the application of the GDT model to the scenario of two robots missioned to clean garbage on the surface of Mars.

The reader is referred to *Specifying, Verifying and Implementing a MAS: a case study* by Mermet, Saval, Simon and Zanuttini, currently submitted to AAMAS'07, for a presentation of the application.

On notation For sake of brevity, when giving the context of a node or leaf, we only give the part of it which is useful to the proof of the leaf or node. Thus if C is the context given for a node, the full one is of the form $C \wedge C'$.

2 Environment

Constants These are variables whose value cannot be modified by any agent or by the environment itself.

- $xMin, yMin, xMax, yMax$ are integers denoting the minimum and maximum x and y coordinates on the grid; it is assumed that $xMin < xMax$ and $yMin < yMax$ hold;
- $size$ is an integer denoting the size (number of cells) of the grid; thus $size = (xMax - xMin + 1) \times (yMax - yMin + 1)$ holds;
- x_{R_2}, y_{R_2} denote the position of R_2, on the x and the y axis, respectively; $xMin \leq x_{R_2} \leq xMax$ and $yMin \leq y_{R_2} \leq yMax$ hold. We also assume $(x_{R_2}, y_{R_2}) \neq (xMin, yMin)$; this is only a technical condition, which can be removed with slightly modifying R_1's GDT.

*GREYC, University of Caen, Boulevard du Maréchal Juin, F-14032 Caen Cedex, France

Variables

- for all x, y with $xMin \leq x \leq xMax$ and $yMin \leq y \leq yMax$, $G(x, y)$ is a Boolean; it is true if and only if there is a piece of garbage on cell (x, y) of the grid;
- $G'(x, y)$ indicates whether there is a piece of garbage on the grid on cell (x, y) at the next moment; note the difference with $G'(x', y')$ (the same remarks apply to $G^{tmp}(x, y)$).

Invariant The system invariant is simply *true*. As a consequence, we ignore it in the following.

Shorthand notation This notation does not define new variables or constants, and is only introduced for sake of brevity and clarity.

- pos_{R_2} is a shorthand for (x_{R_2}, y_{R_2});
- $posMin$ is a shorthand for $(xMin, yMin)$;
- $endGrid$ is a shorthand for $(xMin, yMax)$ if $(yMax - yMin)\%2 = 1$, and for $(xMax, yMax)$ otherwise.
- $(x, y) \prec (x', y')$ (read "(x, y) is before (x', y')") is a shorthand notation for:
$$\left\{ \begin{array}{ll} & y < y' \\ \vee & (y = y' \wedge (y - yMin)\%2 = 0 \wedge x < x') \\ \vee & (y = y' \wedge (y - yMin)\%2 = 1 \wedge x > x') \end{array} \right.$$
- $grid$ ("all cells in the grid except R_2's") is a shorthand notation for $\{(x, y) \mid xMin \leq x \leq xMax \wedge yMin \leq y \leq yMax \wedge (x, y) \neq pos_{R_2}\}$.

3 Robot R_1

We first describe the robot, then give its GDT and finally prove its validity (numbered subsections).

Internal variables

- x, y are integers denoting the current position of R_1 (on the x and y axis, repectively);
- $xSaved, ySaved$ are integers used to save the position where the last piece of garbage has been found (in order to be able to continue searching from this position on);
- $nbAttempts$ is an integer denoting the number of times R_1 has tried to pick the piece of garbage on the current cell;

- *busy* is a boolean; it is true if and only if R_1 is currently carrying a piece of garbage;
- *clean* is the number of cells R_1 has cleaned or observed to be clean.

Invariant R_1's invariant is the following:

$$I_{R_2} = \begin{cases} & xMin \leq x \leq xMax \wedge yMin \leq y \leq yMax \\ \wedge & xMin \leq xSaved \leq xMax \wedge yMin \leq ySaved \leq yMax \\ \wedge & 0 \leq nbAttempts \leq 3 \\ \wedge & clean \subseteq grid \end{cases}$$

Actions

- `initPick` (initialize a series of attempts to pick a piece of garbage):
  - preconditions: $\neg busy \wedge G(x, y)$,
  - postconditions: $nbAttempts' = 0$.
- `pick` (pick the piece of garbage on the current cell):
  - preconditions: $\neg busy \wedge G(x, y) \wedge nbAttempts < 3$,
  - postconditions:

$$\begin{cases} & nbAttempts' = nbAttempts + 1 \\ \wedge & nbAttempts' = 3 \rightarrow \begin{bmatrix} & busy' \\ \wedge & \neg G'(x', y') \\ \wedge & clean' = clean \cup \{pos\} \end{bmatrix} \end{cases}$$

- `drop` (drop the piece of garbage currently held to the current cell):
  - preconditions: $busy \wedge \neg G(x, y)$,
  - postconditions: $\neg busy \wedge G(x, y)$.
- `moveH`(*step*) (move one cell horizontally of given step):
  - preconditions: $step = +1 \vee step = -1$,
  - postconditions: $x' = x + step$.
- `moveV`(*step*) (move one cell vertically into given direction):
  - preconditions: $step = +1 \vee step = -1$,
  - postconditions: $y' = y + step$.
- `recordCleanCell` (record the fact that the current cell is clean):
  - preconditions: $\neg G(x, y)$,
  - postconditions: $clean' = clean \cup \{pos\}$.
- `skip` (do nothing):
  - preconditions: *true*,
  - postconditions: *true*.

Shorthand notation

- dx is a shorthand for $|x' - x|$ and similarly for dy.
- dxR_2 is a shorthand for $|x - x_{R_2}|$, and similarly for dyR_2, $dxMin$, $dyMin$, $dxSaved$, $dySaved$.
- dxR_2' is a shorthand for $|x' - x_{R_2}|$, and similarly for dyR_2', $dxMin'$, $dyMin'$, $dxSaved'$ and $dySaved'$; observe that only x (y) is primed.
- pos is a shorthand for (x, y), pos' is a shorthand for (x', y'), pos^{tmp} is a shorthand for (x^{tmp}, y^{tmp}), and similarly for pos_{R_2}, $posSaved$, $posSaved'$ and $posSaved^{tmp}$.
- $lt(pos)$ ("number of cells before pos and different from R_2's cell") is a shorthand notation for $|\{pos_2 \mid pos_2 \prec pos \wedge pos_2 \neq pos_{R_2}\}|$, i.e., for:

  $$(y - yMin) \times (xMax - xMin + 1) + xMax - x - afterR_2$$

  if $(y - yMin)\%2 = 1$, and for:

  $$(y - yMin) \times (xMax - xMin + 1) + x - xMin - afterR_2$$

  if $(y - yMin)\%2 = 0$, where $afterR_2 = 1$ if $pos_{R_2} \prec (x, y)$ and $afterR_2 = 0$ otherwise.

GDT R_1's GDT is given on Figure 1. It has the following properties:

- Trigerring context: $TC_{R_1} = true$;
- Precondition: $PrecGDT_{R_1} = (clean = \emptyset \wedge \neg busy \wedge pos = posMin)$;
- Initialization $\texttt{init}_{\texttt{R}_1}$:

$$\begin{cases} clean = \emptyset; \\ busy = false; \\ pos = posMin; \\ posSaved = posMin; \\ nbAttempts = 0; \end{cases}$$

Obviously, $\texttt{init}_{\texttt{R}_1}$ establishes $PrecGDT_{R_1}$ and I_{R_2}. Note that R_1 is intended to execute its GTD only once: after executing its GDT, its precondition is not true any more.

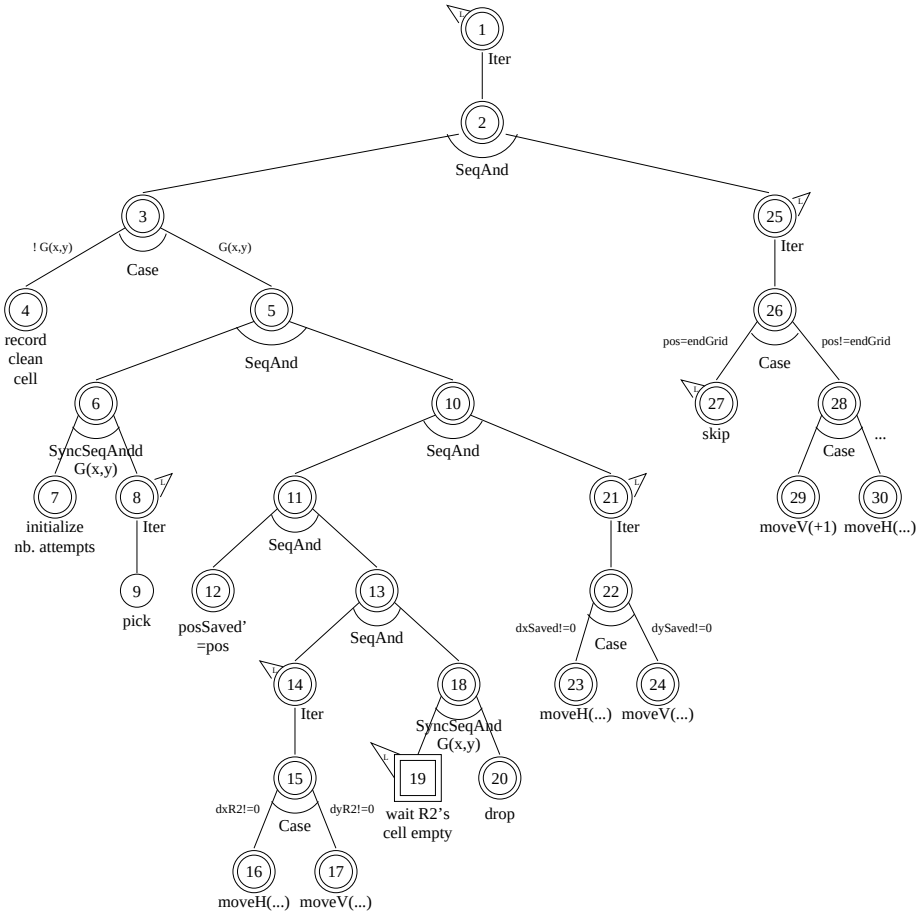

Figure 1: R_1's GDT

3.1 Clean Mars

Visit each cell except R_2's and ensure it is clean or clean it.

Context: $clean = \emptyset \wedge \neg busy \wedge pos = posMin \wedge pos \neq pos_{R_2}$ (from $PrecGDT_{R_1}$ and assumption $pos_{R_2} \neq posMin$)
S.C.: $clean = grid$
Type: Lazy, NS
Operator: Iter
Child(ren): Clean current cell and go to next one (3.2, p. 6)

Proof: Let the variant V be $|grid \setminus clean|$, and let its lower bound be $V_0 = 0$.

We have to prove:

$$\begin{array}{rcl} I_{R_2} \wedge (C_{3.1} \vee C_{3.2}) \wedge \neg SC_{3.1} \wedge T(SC_{3.2}) & \models & T(V) = V_0 \rightarrow T(SC_{3.1}) \\ I_{R_2} \wedge (C_{3.1} \vee C_{3.2}) \wedge \neg SC_{3.1} \wedge \neg T(SC_{3.1}) & \models & T(SC_{3.2}) \rightarrow T(V) < V \\ I_{R_2} \wedge (C_{3.1} \vee C_{3.2}) \wedge \neg SC_{3.1} \wedge \neg T(SC_{3.1}) & \models & T(SC_{3.2}) \rightarrow T(C_{3.2}) \end{array}$$

The first entailment holds because I_{R_2} entails $clean' \subseteq grid$ and $T(V) = V_0$ is $|grid \setminus clean'| = 0$, thus together they entail $clean' = grid$, which is $T(SC_{3.1})$.

As for the second entailment, observe that $T(SC_{3.2})$ entails $clean' = clean \cup \{pos\} \vee clean' = grid$. Thus $\neg T(SC_{3.1}) \wedge T(SC_{3.2})$ entail $clean' = clean \cup \{pos\}$. Together with $C_{3.1}$ this entails $clean = \emptyset \subset \{pos\} = clean'$ and thus $T(V) < V$. Now $C_{3.2}$ entails $clean = lt(pos)$ and thus $pos \notin clean$, from what it follows $clean \subset clean'$ again.

As for the third entailment, observe that $T(SC_{3.2})$ entails $T(C_{3.2}) \vee (\neg busy' \wedge clean' = grid)$. Thus $\neg T(SC_{3.1}) \wedge T(SC_{3.2})$ entail $T(C_{3.2})$.

3.2 Clean current cell and go to next one

Ensure the current cell is clean or clean it, and move to the next one different from R_2's, if any.

Context: $\neg busy \wedge clean = lt(pos) \wedge pos \neq pos_{R_2}$ (from $C_{3.1}$)
S.C.: $\left\{ \begin{array}{ll} & (\neg busy' \wedge pos' \neq pos_{R_2} \wedge clean' = clean \cup \{pos\} \wedge clean' = lt(pos')) \\ \vee & (\neg busy' \wedge clean' = grid) \end{array} \right.$
Type: Non lazy, NS
Operator: SeqAnd
Child(ren):
- Clean current cell (3.3, p. 6)
- Go to next cell different from R_2's (3.25, p. 17)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.2} \models [v' = v_{tmp}]T((SC_{3.3})_i) \wedge [v = v_{tmp}]T(SC_{3.25}) \rightarrow T(SC_{3.2})$$

Since $x, y, clean$ are internal variables, $[v = v_{tmp}]SC_{3.25}$ entails either $\neg busy' \wedge clean' = grid$ or $\neg busy' \wedge pos' \neq pos_{R_2} \wedge clean' = clean^{tmp} \wedge clean' = lt(pos')$. In the first case, this is exactly the second disjunct of $T(SC_{3.2})$. In the second case, since $[v' = v_{tmp}]T((SC_{3.3})_i)$ entails $clean^{tmp} = clean \cup \{pos\}$, it is easily seen that the first conjunct of $T(SC_{3.2})$ is entailed as well. Thus in both cases, $T(SC_{3.2})$ is entailed.

3.3 Clean current cell

Ensure the current cell is clean or clean it.

Context: $\neg busy \wedge clean = lt(pos) \wedge pos \neq pos_{R_2}$ (from $C_{3.2}$)
S.C.: $clean' = clean \cup \{pos\} \wedge \neg busy' \wedge pos' \neq pos_{R_2} \wedge clean' = lt(pos') \cup \{pos'\}$
Type: Non lazy, NS
Operator: Case
Child(ren):
- (If $\neg G(x, y)$) Record current cell is clean (3.4, p. 7)
- (If $G(x, y)$) Clean current cell (3.5, p. 7)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.3} \models \begin{cases} condition_1 \vee condition_2 \\ condition_1 \wedge T(SC_{3.4}) \rightarrow T(SC_{3.3}) \\ condition_2 \wedge T(SC_{3.5}) \rightarrow T(SC_{3.3}) \end{cases}$$

where $condition_1 = \neg G(x, y)$ and $condition_2 = G(x, y)$.

The first entailment obviously holds since $condition_1 = \neg condition_2$. The second one holds because $SC_{3.4}$ entails $clean' = clean \cup \{pos\}$ and $\neg busy'$ directly, $clean' = lt(pos') \cup \{pos'\}$ together with $clean = lt(pos)$ in $C_{3.3}$ and $pos' \neq pos_{R_2}$ together with $pos \neq pos_{R_2}$ in $C_{3.3}$. The third one holds for exactly the same reasons since $SC_{3.5} = SC_{3.4}$.

3.4 Record current cell is clean (leaf)

Record the fact that the current cell is clean.

Context: $\neg G(x, y) \wedge \neg busy \wedge clean = lt(pos) \wedge pos \neq pos_{R_2}$ (from condition on case branch and $C_{3.3}$)
S.C.: $clean' = clean \cup \{pos\} \wedge \neg busy' \wedge pos' = pos$
Type: Non lazy, NS
Action: `recordCleanCell`;

Proof: Clearly, $C_{3.4}$ entails the preconditions of `recordCleanCell`. The postconditions of `recordCleanCell` entail $clean' = clean \cup \{pos\}$. Finally, since *busy* and *pos* are not modified by `recordCleanCell`, $\neg busy'$ follows from $C_{3.4}$ and $pos' = pos$ is obvious.

Invariant: Obvious for variables $x, y, xSaved, ySaved, nbAttempts$ since they are not modified by `recordCleanCell`. Thus we are left with proving $clean' \subseteq grid$, which amounts to proving $pos \in grid$ since I_{R_2} entails $clean \subseteq grid$ and $T(SC_{3.4})$ entails $clean' = clean \cup \{pos\}$. But from I_{R_2} it follows $xMin \leq x \leq xMax$ and $yMin \leq y \leq yMax$. Since $C_{3.4}$ entails $pos \neq pos_{R_2}$, we finally have $pos \in grid$, as desired.

3.5 Clean current cell

Clean the current cell.

Context: $G(x,y) \wedge \neg busy \wedge clean = lt(pos) \wedge pos \neq pos_{R_2}$ (from condition on case branch and $C_{3.3}$)
S.C.: $clean' = clean \cup \{pos\} \wedge \neg busy' \wedge pos' = pos$
Type: Non lazy, NS
Operator: SeqAnd
Child(ren):
- Pick (3.6, p. 8)
- Get rid (3.10, p. 10)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.3} \models [v' = v_{tmp}]T((SC_{3.6})_i) \wedge [v = v_{tmp}]T(SC_{3.10}) \rightarrow T(SC_{3.3})$$

Since $clean, x, y$ are internal variables, $[v' = v_{tmp}](SC_{3.6})_i$ entails $clean^{tmp} = clean \cup \{pos\}$ and $pos^{tmp} = pos$. Now $[v = v_{tmp}]SC_{3.10}$ entails $\neg busy'$, $pos' = pos^{tmp}$ and $clean' = clean^{tmp}$, thus $T(SC_{3.5})$ follows.

3.6 Pick

Pick the piece of garbage on the current cell.

Context: $G(x,y) \wedge \neg busy \wedge clean = lt(pos) \wedge pos \neq pos_{R_2}$ (from $C_{3.5}$)
S.C.: $\neg G'(x',y') \wedge busy' \wedge clean' = clean \cup \{pos\} \wedge pos' = pos$
Type: Non lazy, NS
Operator: SyncSeqAnd($G(x,y)$)
Child(ren):
- Initialize attempts (3.7, p. 8)
- Try to pick until success (3.8, p. 9)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.6} \models [v' = v_{tmp}]T((SC_{3.7})_{i,G(x,y)}) \wedge [v = v_{tmp}]T(SC_{3.8}) \rightarrow T(SC_{3.6})$$

Since $clean, pos$ are internal variables, $[v' = v_{tmp}](T(SC_{3.7}))_i$ entails $clean^{tmp} = clean$ and $pos^{tmp} = pos$. Then it is easily that together with $[v = v_{tmp}]T(SC_{3.8})$ this entails $T(SC_{3.6})$.

3.7 Initialize attempts (leaf)

Initialize the number of attempts to pick a piece of garbage.

Context: $G(x,y) \wedge \neg busy \wedge clean = lt(pos) \wedge pos \neq pos_{R_2}$ (from $C_{3.6}$)

S.C.: $\begin{cases} & nbAttempts = 0 \\ \wedge & clean' = clean \wedge clean' = lt(pos') \\ \wedge & \neg busy \\ \wedge & G(x,y) \\ \wedge & pos' = pos \wedge pos' \neq pos_{R_2} \end{cases}$
Type: Non lazy, NS
Action: `initPick`;

Proof: Obvious since the preconditions of `initPick` are entailed by $C_{3.7}$, conjunct $nbAttempts' = 0$ in $T(SC_{3.7})$ is the postcondition of `initPick`, and every other conjunct in $T(SC_{3.7})$ is obtained from the context together with the fact that the action does not affect them (recall that environment variables are assumed not to change values during the execution of an elementary goal).

Invariant: Obvious for variables $x, y, xSaved, ySaved, clean$, since they are not modified. Obvious for variable $nbAttempts$ too since it is set to 0.

3.8 Try to pick until success

Try to pick the piece of garbage on the current cell until success.

Context: $nbAttempts = 0 \wedge G(x,y) \wedge \neg busy \wedge clean = lt(pos) \wedge pos \neq pos_{R_2}$ (from $SC_{3.7}$ since $G(x,y)$ is synchronized in 3.6)
S.C.: $\neg G'(x',y') \wedge busy' \wedge clean' = clean \cup \{pos\} \wedge pos' = pos$
Type: Lazy, NS
Operator: Iter
Child(ren): Try to pick (3.9, p. 10)

Proof: Let V be the variant $3 - nbAttempts$, and let its lower bound be $V_0 = 0$. Observe that V is well-defined, since I_{R_2} entails $nbAttempts \leq 3$. We have to prove:

$$\begin{array}{rcl} I_{R_2} \wedge \neg SC_{3.8} \wedge (C_{3.8} \vee C_{3.9}) \wedge T(SC_{3.9}) & \models & T(V) = V_0 \rightarrow T(SC_{3.8}) \\ I_{R_2} \wedge \neg SC_{3.8} \wedge \neg T(SC_{3.8}) \wedge (C_{3.8} \vee C_{3.9}) & \models & T(SC_{3.9}) \rightarrow T(V) < V \\ I_{R_2} \wedge \neg SC_{3.8} \wedge \neg T(SC_{3.8}) \wedge (C_{3.8} \vee C_{3.9}) & \models & T(GPF_{3.9}) \rightarrow T(V) < V \\ I_{R_2} \wedge \neg SC_{3.8} \wedge (C_{3.8} \vee C_{3.9}) \wedge \neg T(SC_{3.8}) & \models & T(SC_{3.9}) \rightarrow T(C_{3.9}) \\ I_{R_2} \wedge \neg SC_{3.8} \wedge (C_{3.8} \vee C_{3.9}) \wedge \neg T(SC_{3.8}) & \models & T(GPF_{3.9}) \rightarrow T(C_{3.9}) \end{array}$$

The first entailment is obvious since $T(SC_{3.9})$ is exactly $T(SC_{3.8})$.

The second entailment holds because $\neg T(SC_{3.8}) \wedge T(SC_{3.9})$ is false. The third one holds because $T(GPF_{3.9})$ entails $nbAttempts' = nbAttempts + 1$, which in turn entails $3 - nbAttempts' = 3 - nbAttempts - 1 < 3 - nbAttempts$, that is, $T(V) < V$.

The fourth entailment is obvious since again $T(SC_{3.9})$ is not consistent with $\neg T(SC_{3.8})$. Finally, the fifth one holds because $T(GPF_{3.9})$ together with either $C_{3.8}$ or with $C_{3.9}$ clearly entails $T(C_{3.9})$.

3.9 Try to pick (leaf)

Try to pick the piece of garbage on the current cell.

Context: $G(x,y) \wedge \neg busy \wedge clean = lt(pos) \wedge pos \neq pos_{R_2} \wedge nbAttempts < 3$ (from $C_{3.8}$)
S.C.: $busy' \wedge \neg G'(x',y') \wedge clean' = clean \cup \{pos\} \wedge pos' = pos$
G.P.F.: $\begin{cases} & nbAttempts' = nbAttempts + 1 \wedge nbAttempts' < 3 \\ \wedge & pos' = pos \wedge clean' = clean \wedge busy' = busy \\ \wedge & G(',x')g \end{cases}$
Type: Non lazy, NNS
Action: `pick`;

Proof: Obvious.

G.P.F.: Obvious.

Invariant: Obvious for variables $x, y, xSaved, ySaved$, since they are not modified. As for variable *clean*, the proof is the same as that in Goal 3.4 on page 7, since $pos \neq pos_{R_2}$ is entailed by $C_{3.9}$ too. Finally, $nbAttempts' \leq 3$ holds because $C_{3.9}$ entails $nbAttempts < 3$ and `initPick` has $nbAttempts' = nbAttempts + 1$ as a postcondition, and $0 \leq nbAttempts'$ holds because I_{R_2} entails $0 \leq nbAttempts$ and $nbAttempts' = nbAttempts$.

3.10 Get rid

Bring the piece or garbage currently held to R_2 and come back to the current cell.

Context: *busy* (from $SC_{3.6}$)
S.C.: $\neg busy' \wedge pos' = pos \wedge clean' = clean$
Type: Non lazy, NS
Operator: SeqAnd
Child(ren):
- Save position and give R_2 (3.11, p. 11)
- Go to saved position (3.21, p. 15)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.10} \models [v' = v_{tmp}]T((SC_{3.11})_i) \wedge [v = v_{tmp}]T(SC_{3.21}) \rightarrow T(SC_{3.10})$$

Since $x, y, xSaved, ySaved$ are internal variables, $[v' = v_{tmp}](T(SC_{3.11}))_i$ entails $posSaved^{tmp} = pos$; on the other hand, $[v = v_{tmp}]T(SC_{3.21})$ entails $pos' = posSaved^{tmp}$ and $\neg busy'$, and thus conjuncts $\neg busy'$ and $pos' = pos$ in $T(SC_{3.10})$ follow. Finally, $clean' = clean$ is entailed because variable *clean* is internal and does not occur in the subtree rooted at this node.

3.11 Save position and give R_2

Save the current position and bring the piece of garbage currently held to R_2.

Context: *busy* (from $C_{3.10}$)
S.C.: $pos' = pos_{R_2} \wedge \neg busy' \wedge G'(x', y') \wedge posSaved' = pos$
Type: Non lazy, NS
Operator: SeqAnd
Child(ren):
- Save position (3.12, p. 11)
- Go and give R_2 (3.13, p. 11)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.11} \models [v' = v_{tmp}]T((SC_{3.12})_i) \wedge [v = v_{tmp}]T(SC_{3.13}) \rightarrow T(SC_{3.11})$$

Conjuncts $pos' = pos_{R_2}$, $\neg busy'$ and $G'(x', y')$ in $T(SC_{3.11})$ follow directly from $[v' = v_{tmp}]T(SC_{3.13})$. As for conjunct $posSaved' = pos$, it follows from $posSaved^{tmp} = pos^{tmp} \wedge pos^{tmp} = pos$, which is entailed by $[v' = v_{tmp}](SC_{3.12})_i$, together with $posSaved' = pos$, which is entailed by $[v = v_{tmp}]T(SC_{3.13})$.

3.12 Save position (leaf)

Save the current position into variables $xSaved, ySaved$ in order to search for garbage from the last visited cell on next time.

Context: *busy* (from $C_{3.11}$)
S.C.: $posSaved' = pos' \wedge pos' = pos \wedge busy'$
Type: Non lazy, NS
Action: $posSaved' = pos$;

Proof: Obvious.

Invariant: Obvious for variables *clean* and *nbAttempts*, which is not modified by the action. Also obvious for x, y, which are not modified by the action either. Thus $xMin \leq x' \leq xMax$ and $yMin \leq y' \leq yMax$ hold. Since $SC_{3.12}$ is $(xSaved', ySaved') = (x, y)$, we also have $xMin \leq xSaved' \leq xMax$ and $yMin \leq ySaved' \leq yMax$.

3.13 Go and give R_2

Go to R_2's cell and give it the piece of garbage currently held.

Context: $posSaved = pos \wedge busy$ (from $SC_{3.12}$)
S.C.: $pos' = pos_{R_2} \wedge \neg busy' \wedge G'(x', y') \wedge posSaved' = posSaved$

Type: Non lazy, NS
Operator: SeqAnd
Child(ren):
- Go to R_2 (3.14, p. 12)
- Give to R_2 (3.18, p. 14)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.13} \models [v' = v_{tmp}]T((SC_{3.14})_i) \wedge [v = v_{tmp}]T(SC_{3.18}) \rightarrow T(SC_{3.13})$$

Since *pos* is internal, $[v' = v_{tmp}](T(SC_{3.14}))_i$ entails $pos^{tmp} = pos_{R_2}$, which together with $pos' = pos^{tmp}$ in $[v' = v_{tmp}]SC_{3.18}$ entails $pos' = pos_{R_2}$. Now $\neg busy'$ and $G'(x', y')$ are entailed directly by $[v = v_{tmp}]SC_{3.18}$, and $posSaved' = posSaved$ is entailed by the fact that variables $xSaved, ySaved$ are internal and do not occur in the subtree rooted at this node.

3.14 Go to R_2

Go to R_2's cell.

Context: *busy* (from $C_{3.13}$)
S.C.: $pos = pos_{R_2} \wedge busy$
Type: Lazy, NS
Operator: Iter
Child(ren): Move towards R_2 (3.15, p. 13)

Proof: Since *busy* is internal and does not occur in the subtree rooted at this node, its value obviously propagates through this node, so we ignore it in the following.

Let V be the variant $dxR_2 + dyR_2$, and let its lower bound be $V_0 = 0$. Then we have to prove:

$$\begin{array}{rcl} I_{R_2} \wedge (C_{3.14} \vee C_{3.15}) \wedge \neg SC_{3.14} \wedge T(SC_{3.15}) & \models & T(V) = V_0 \rightarrow T(SC_{3.14}) \\ I_{R_2} \wedge (C_{3.14} \vee C_{3.15}) \wedge \neg SC_{3.14} \wedge \neg T(SC_{3.14}) & \models & T(SC_{3.15}) \rightarrow T(V) < V \\ I_{R_2} \wedge (C_{3.14} \vee C_{3.15}) \wedge \neg SC_{3.14} \wedge \neg T(SC_{3.14}) & \models & T(SC_{3.15}) \rightarrow T(C_{3.15}) \end{array}$$

The first entailment holds because $T(V) = V_0$ entails $x' = x_{R_2}$ and $y' = y_{R_2}$ and thus $pos' = pos_{R_2}$, that is, $T(SC_{3.14})$.

The second entailment holds because each disjunct in $T(SC_{3.15})$ entails $dxR'_2 + dyR'_2 = dxR_2 + dyR_2 - 1$ and thus $dxR'_2 + dyR'_2 < dxR_2 + dyR_2$, that is, $T(V) < V$.

Finally, the third entailment holds because $\neg T(SC_{3.14})$ is $pos' \neq pos_{R_2}$, i.e., $T(C_{3.15})$.

3.15 Move towards R_2

Move one cell towards R_2's position.

Context: $pos \neq pos_{R_2}$ (from $\neg SC_{3.14}$ since 3.14 is lazy)
S.C.: $\begin{cases} & (dxR_2' = dxR_2 - 1 \wedge dyR_2' = dyR_2) \\ \vee & (dxR_2' = dxR_2 \wedge dyR_2' = dyR_2 - 1) \end{cases}$
Type: Non lazy, NS
Operator: Case
Child(ren):
- (If $dxR_2 \neq 0$) Move towards R_2 along x (3.16, p. 13)
- (If $dyR_2 \neq 0$) Move towards R_2 along y (3.17, p. 14)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.15} \models \begin{cases} condition_1 \vee condition_2 \\ condition_1 \wedge T(SC_{3.16}) \rightarrow T(SC_{3.15}) \\ condition_2 \wedge T(SC_{3.17}) \rightarrow T(SC_{3.15}) \end{cases}$$

where $condition_1 = (dxR_2 \neq 0)$ and $condition_2 = (dyR_2 \neq 0)$.

Since $C_{3.15}$ entails $pos \neq pos_{R_2}$, obviously the first entailment holds. The other two entailments are obvious as well, considering only $SC_{3.16}$ and $SC_{3.17}$.

3.16 Move towards R_2 along x (leaf)

Move one cell towards R_2's position along the x axis.

Context: $dxR_2 \neq 0$ (from condition on case branch)
S.C.: $dxR_2' = dxR_2 - 1 \wedge y' = y$
Type: Non lazy, NS
Action: `moveH`$(-\frac{x - x_{R_2}}{|x - x_{R_2}|})$;

Proof: Observe that $\frac{x - x_{R_2}}{|x - x_{R_2}|}$ is well-defined since $C_{3.16}$ entails $|x - x_{R_2}| \neq 0$. Moreover, obviously the preconditions of `moveH`(.) are satisfied. Now we have to prove that its postconditions entail $T(SC_{3.16})$. Since $y' = y$ is obviously entailed and $C_{3.16}$ entails $|x - x_{R_2}| \neq 0$, it is enough to prove:

$$|x - x_{R_2}| \neq 0 \wedge x' = x - \frac{x - x_{R_2}}{|x - x_{R_2}|} \models |x' - x_{R_2}| = |x - x_{R_2}| - 1$$

Assume $x > x_{R_2}$; then $|x - x_{R_2}| = x - x_{R_2}$ and $|x - 1 - x_{R_2}| = x - x_{R_2}$, and thus we are left with proving that $x - x_{R_2} \neq 0 \wedge x' = x - 1$ entails $x' - x_{R_2} = x - x_{R_2} - 1$, which is obviously true. The proof is dual with the assumption $x < x_{R_2}$, and with the assumption $x = x_{R_2}$ the entailment is true because the premisses are false (since $C_{3.16}$ entails $x \neq x_{R_2}$).

Invariant: Obvious for variables $y, xSaved, ySaved, clean, nbAttempts$, since they are not modified by the action. We thus consider variable x.

Assume $x > x_{R_2}$. Then $x' = x - 1$, as shown above, and thus $x' \geq x_{R_2}$. Now I_{R_2} entails $x \leq xMax$, and $xMin \leq x_{R_2}$ is always true. It follows $xMin \leq x_{R_2} \leq x' < x \leq xMax$ and thus $xMin \leq x' \leq xMax$. The proof for the case $x < x_{R_2}$ is dual, and the case $x = x_{R_2}$ yields false premisses as above.

3.17 Move towards R_2 along y (leaf)

Move one cell towards R_2's position along the y axis.

Context: $dyR_2 \neq 0$ (from condition on case branch)
S.C.: $dyR_2' = dyR_2 - 1 \wedge x' = x$
Type: Non lazy, NS
Action: `moveV`$(-\frac{y - y_{R_2}}{|y - y_{R_2}|})$;

Proof: Similar to the proof of leaf 3.16 on page 13.

Invariant: Idem.

3.18 Give to R_2

Give the piece of garbage currently held to R_2.

Context: $pos = pos_{R_2} \wedge busy$ (from $SC_{3.14}$)
S.C.: $\neg busy' \wedge G'(x', y') \wedge pos' = pos$
Type: Non lazy, NS
Operator: SyncSeqAnd$(G(x, y))$
Child(ren):
- Wait for R_2's cell to be empty (3.19, p. 14)
- Drop (3.20, p. 15)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.18} \models [v' = v_{tmp}]T((SC_{3.19})_{i,G(x,y)}) \wedge [v = v_{tmp}]T(SC_{3.20}) \rightarrow T(SC_{3.18})$$

Conjuncts $\neg busy'$ and $G'(x', y')$ are entailed directly by $[v = v_{tmp}]T(SC_{3.20})$, and $pos' = pos$ holds because variable pos is internal and does not occur in the subtree rooted at this node.

3.19 Wait for R_2's cell to be empty (external)

Wait for R_2's cell to be empty.

Context: $pos = pos_{R_2} \wedge busy$ (from $C_{3.18}$)
S.C.: $\neg G(x, y) \wedge busy$
Type: Lazy, NS
Responsible: R_2, Clean cell (4.1, p. 21)

Proof: We first have to check that the referenced node is necessarily satisfiable, which is indeed true.

Now we have to show that the achievement of Node 4.1 of R_2 entails the achievement of this node. Indeed, since *pos* and *busy* are internal we have $pos' = pos$ and $busy' = busy$, and thus: $C_{3.19} \wedge T(SC_{4.1}) \models \neg G'(x_{R_2}, y_{R_2}) \wedge pos' = pos_{R_2} \wedge busy' \models \neg G'(x', y') \wedge busy'$ which is exactly $T(SC_{3.19})$.

Finally, we have to prove that every node of R_2's GDT whose context is compatible with $C_{3.19}$ eventually leads to the achievement of its Node 4.1, and that Node 4.1 is indeed compatible. This latter point is clearly true. Now since this Node 3.19 is lazy, we have to check this point for every node in R_2's GDT whose context is compatible with $C_{3.19} \wedge \neg SC_{3.19}$, that is, $pos = pos_{R_2} \wedge G(x, y)$. Clearly, only Nodes 4.1 and 4.2 are compatible with this context, and obviously both eventually end up with satisfaction of $SC_{4.1}$. Finally, if R_2 is not executing its GDT, then since its precondition is true (as its satisfaction condition shows), $G(x, y)$ should have triggerred its GDT, a contradiction.

3.20 Drop (leaf)

Drop the piece of garbage currently held onto the current cell.

Context: $\neg G(x, y) \wedge busy$ (from $SC_{3.19}$ since $G(x, y)$ is synchronized in 3.18)
S.C.: $\neg busy' \wedge G'(x', y')$
Type: Non lazy, NS
Action: `drop`;

Proof: Clearly, the preconditions of `drop` are satisfied, and its postconditions entail $SC_{3.20}$.

Invariant: Obvious since no variable which occurs in the invariant is modified by `drop`.

3.21 Go to saved position

Go back to the saved position.

Context: $\neg busy$ (from $SC_{3.11}$)
S.C.: $pos = posSaved \wedge \neg busy$
Type: Lazy, NS

Operator: Iter
Child(ren): Move towards saved position (3.22, p. 16)

Proof: Similar to that of node 3.14 on page 12, replacing dxR_2 with $dxSaved$, x_{R_2} with $xSaved$, pos_{R_2} with $posSaved$ and $busy$ with $\neg busy$.

3.22 Move towards saved position

Move one cell towards the saved position.

Context: $pos \neq posSaved$ (from $\neg SC_{3.21}$ since 3.21 is lazy)
S.C.: $\begin{cases} & (dxSaved' = dxSaved - 1 \wedge dySaved' = dySaved) \\ \vee & (dxSaved' = dxSaved \wedge dySaved' = dySaved - 1) \end{cases}$
Type: Non lazy, NS
Operator: Case
Child(ren):
- (If $dxSaved \neq 0$) Move towards saved position along (3.23, p. 16)
- (If $dySaved \neq 0$) Move towards saved position along y (3.24, p. 16)

Proof: Similar to the proof of node 3.15 on page 13, replacing dxR_2 with $dxSaved$, x_{R_2} with $xSaved$ and pos_{R_2} with $posSaved$.

3.23 Move towards saved position along (leaf)

Move one cell towards the saved position along the x axis.

Context: $dxSaved \neq 0$ (from condition on case branch)
S.C.: $dxSaved' = dxSaved - 1 \wedge y' = y$
Type: Non lazy, NS
Action: `moveH`$(-\frac{x - xSaved}{|x - xSaved|})$;

Proof: Similar to the proof of leaf 3.16 on page 13, replacing dxR_2 with $dxSaved$ and x_{R_2} with $xSaved$.

Invariant: Idem since the invariant guarantees $xSaved \geq xMin$.

3.24 Move towards saved position along y (leaf)

Move one cell towards the saved position along the y axis.

Context: $dySaved \neq 0$ (from condition on case branch)
S.C.: $dySaved' = dySaved - 1 \wedge x' = x$

Type: Non lazy, NS
Action: `moveV`$(-\frac{y-ySaved}{|y-ySaved|})$;

Proof: Similar to the proof of leaf 3.23 on page 16.

Invariant: Idem.

3.25 Go to next cell different from R_2's

Move to the next cell different from R_2's and stop if there is none.

Context: $\neg busy \wedge clean = lt(pos) \cup \{pos\} \wedge pos \neq pos_{R_2}$ (from $SC_{3.3}$)
S.C.: $\begin{cases} & \neg busy' \wedge pos' \neq pos_{R_2} \wedge clean' = clean \wedge clean' = lt(pos') \\ \vee & \neg busy' \wedge clean' = grid \end{cases}$
Type: Lazy, NS
Operator: Iter
Child(ren): Go to next cell or stop (3.26, p. 18)

Proof: We have to prove:

$$\begin{array}{rcl} I_{R_2} \wedge (C_{3.25} \vee C_{3.26}) \wedge \neg SC_{3.25} \wedge T(SC_{3.26}) & \models & T(V) = V_0 \rightarrow T(SC_{3.25}) \\ I_{R_2} \wedge (C_{3.25} \vee C_{3.26}) \wedge \neg SC_{3.25} \wedge \neg T(SC_{3.25}) & \models & T(SC_{3.26}) \rightarrow T(V) < V \\ I_{R_2} \wedge (C_{3.25} \vee C_{3.26}) \wedge \neg SC_{3.25} \wedge \neg T(SC_{3.25}) & \models & T(SC_{3.26}) \rightarrow T(C_{3.26}) \end{array}$$

Whatever V is, the proof of $\neg busy'$ is obvious for all entailments, since $C_{3.25}$ entails $\neg busy$, $busy$ is internal and does not occur in the subtree rooted at this node. So we do not consider it in the following.

Now define variant V to be an integer equal to:

$$\begin{cases} +1 & \text{if} \quad pos \prec pos_{R_2} \\ 0 & \text{if} \quad pos = pos_{R_2} \\ -1 & \text{if} \quad pos \succ pos_{R_2} \end{cases}$$

Moreover, define its lower bound $V_0 = -1$.

Consider the first entailment. If disjunct $clean' = grid$ in $T(SC_{3.26})$ is true, then $T(SC_{3.25})$ obviously follows. Now if the other disjunct is true, together with $T(V) = V_0$ this entails $pos' \neq pos_{R_2} \wedge clean' = clean$, so we are left with proving $clean' = lt(pos')$. We consider three cases:

- if $pos = pos_{R_2}$ is true, then so is context $C_{3.26}$, since $C_{3.25}$ entails $pos \neq pos_{R_2}$; then $dx + dy = 1 \wedge pos' \succ pos$ entails $lt(pos') = lt(pos)$ and $C_{3.26}$ entails $clean = lt(pos)$, thus together with $clean' = clean$ we get $clean' = lt(pos')$;
- if $pos \neq pos_{R_2}$ and context $C_{3.25}$ are true, then $dx + dy = 1 \wedge pos' \succ pos$ entails $lt(pos') = lt(pos) \cup \{pos\}$ and $C_{3.25}$ entails $clean = lt(pos) \cup \{pos\}$, thus together with $clean' = clean$ we get $clean' = lt(pos')$;

- if $pos \neq pos_{R_2}$ and context $C_{3.26}$ are true, then $dx + dy = 1 \wedge pos' \succ pos$ entails $lt(pos') = lt(pos) \cup \{pos\}$ and $C_{3.26}$ entails $clean = lt(pos) \cup \{pos\}$, thus together with $clean' = clean$ we get $clean' = lt(pos')$.

Now consider the second entailment. The premisses entail $pos' = pos_{R_2}$, for otherwise $T(SC_{3.26})$ would entail $T(SC_{3.25})$ with the same proof as above. Thus $T(V) = 0$. Now since $T(SC_{3.26})$ entails $pos' \succ pos$, we get $pos \prec pos_{R_2}$ and thus $V = 1 > T(V)$.

The proof of the third entailment is similar to that of the first one.

3.26 Go to next cell or stop

Move to the next cell and stop if there is none.

Context: $\left\{ \begin{array}{ll} & pos \neq pos_{R_2} \rightarrow clean = lt(pos) \cup \{pos\} \\ \wedge & pos = pos_{R_2} \rightarrow clean = lt(pos) \end{array} \right.$
(from $C_{3.25}$)
S.C.: $(dx + dy = 1 \wedge pos' \succ pos \wedge clean' = clean) \vee clean' = grid$
Type: Non lazy, NS
Operator: Case
Child(ren):
- (If $(pos = endGrid)$) Stop if end of grid (3.27, p. 18)
- (If $pos \neq endGrid$) Go to next cell (3.28, p. 19)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.26} \models \left\{ \begin{array}{l} condition_1 \vee condition_2 \\ condition_1 \wedge T(SC_{3.27}) \rightarrow T(SC_{3.26}) \\ condition_2 \wedge T(SC_{3.28}) \rightarrow T(SC_{3.26}) \end{array} \right.$$

where $condition_1 = (pos = endGrid)$ and $condition_2 = (pos \neq endGrid)$.

The first implication is obvious, since $condition_1$ is logically equivalent to $\neg condition_2$. The other two implications are obvious too, since $T(SC_{3.27})$ is exactly the second disjunct in $T(SC_{3.26})$ and $T(SC_{3.28})$ is exactly the first one.

3.27 Stop if end of grid (leaf)

Stop if the end of the grid has been reached.

Context: $\left\{ \begin{array}{ll} & pos = endGrid \\ \wedge & pos \neq pos_{R_2} \rightarrow clean = lt(pos) \cup \{pos\} \\ \wedge & pos = pos_{R_2} \rightarrow clean = lt(pos) \end{array} \right.$
(from $C_{3.26}$ and condition on case branch)
S.C.: $clean' = grid$
Type: Lazy, NS
Action: `skip`;

Proof: Observe that $grid = lt(endGrid)$ if $endGrid = pos_{R_2}$ and $grid = lt(endGrid) \cup \{endGrid\}$ otherwise. Now assume $pos = pos_{R_2}$. Then from $C_{3.27}$ it follows $clean = lt(pos)$ and $pos = endGrid$, thus $clean = lt(endGrid) = grid$. Finally, assume $pos \neq pos_{R_2}$. Then from $C_{3.27}$ it follows $clean = lt(pos) \cup \{pos\}$ and $pos = endGrid$, thus $clean = lt(endGrid) \cup \{endGrid\} = grid$.

Invariant: Obvious since no variable is modified by `skip`.

3.28 Go to next cell

Go to the next cell.

Context: $pos \neq endGrid$ (from condition on case branch)
S.C.: $dx + dy = 1 \wedge pos' \succ pos \wedge clean' = clean$
Type: Non lazy, NS
Operator: Case
Child(ren):
- (If $(x = xMax \wedge (y - yMin)\%2 = 0) \vee (x = xMin \wedge (y - yMin)\%2 = 1)$) Change line (3.29, p. 19)
- (If $(x \neq xMax \vee (y - yMin)\%2 = 1) \wedge (x \neq xMin \vee (y - yMin)\%2 = 0)$) Go to next cell on same line (3.30, p. 20)

Proof: We have to prove:

$$I_{R_2} \wedge C_{3.28} \models \begin{cases} condition_1 \vee condition_2 \\ condition_1 \wedge T(SC_{3.29}) \rightarrow T(SC_{3.28}) \\ condition_2 \wedge T(SC_{3.30}) \rightarrow T(SC_{3.28}) \end{cases}$$

where $condition_1$ is $(x = xMax \wedge (y - yMin)\%2 = 0) \vee (x = xMin \wedge (y - yMin)\%2 = 1)$ and $condition_2$ is $(x \neq xMax \vee (y - yMin)\%2 = 1) \wedge (x \neq xMin \vee (y - yMin)\%2 = 0)$.

The first implication is clear, since $condition_1$ is logically equivalent to $\neg condition_2$. The other are obvious too since *clean* is internal and does not occur in the subtree rooted at this node, and $(T(SC_{3.29}) \vee T(SC_{3.30})) \models dx + dy = 1 \wedge pos' \succ pos$ obviously holds.

3.29 Change line (leaf)

Change line.

Context: $\begin{cases} & pos \neq endGrid \\ \wedge & \begin{bmatrix} & (x = xMax \wedge (y - yMin)\%2 = 0) \\ \vee & (x = xMin \wedge (y - yMin)\%2 = 1) \end{bmatrix} \end{cases}$
(from $C_{3.28}$ and condition on case branch)

S.C.: $dy = 1 \wedge dx = 0 \wedge pos' \succ pos$
Type: Non lazy, NS
Action: `moveV`$(+1)$;

Proof: Obvious.

Invariant: Obvious for $x, xSaved, ySaved, clean, nbAttempts$, which are not modified by the action.

Now consider variable y. From I_{R_2} it follows $yMin \leq y$ and thus, $yMin \leq y + 1 = y'$. Now from I_{R_2} it also follows $y \leq yMax$. If $y < yMax$, then $y' = y + 1 \leq yMax$. Finally, if $y = yMax$, then from $C_{3.29}$ it follows $pos = endGrid$ and thus $false$, thus $y \leq yMax$ follows vacuously.

3.30 Go to next cell on same line (leaf)

Move one cell on the same line.

Context: $(x \neq xMax \vee (y - yMin)\%2 = 1) \wedge (x \neq xMin \vee (y - yMin)\%2 = 0)$ (from condition on case branch)
S.C.: $dx = 1 \wedge dy = 0 \wedge pos' \succ pos$
Type: Non lazy, NS
Action: `moveH`$(1 - 2(y - yMin)\%2)$;

Proof: Obvious; observe that if $(y - yMin)\%2 = 0$, then $1 - 2(y - yMin)\%2 = +1$ and if $(y - yMin)\%2 = 1$, then $1 - 2(y - yMin)\%2 = -1$.

Invariant: Obvious for $y, xSaved, ySaved, clean, nbAttempts$, which are not modified by the action.

Now consider variable x. From I_{R_2} it follows $x \leq xMax$. Thus if $x < xMax$ we get $x' = x + 1 < xMax + 1$ or $x' = x - 1 < xMax + 1$, and thus $x' \leq xMax$. Now if $x = xMax$ then from the first conjunct in $C_{3.30}$ it follows $(y - yMin)\%2 = 1$ and thus $x' = x - 1 < x \leq xMax$. The proof of $xMin \leq x$ is dual.

4 Robot R_2

We first describe the robot, then give its GDT and finally prove its validity (numbered subsections).

Internal variables R_2 has only one internal variable, namely a Boolean $busy_{R_2}$. Its value is $true$ if and only if R_2 is currently holding a piece of garbage.

Invariant R_2's invariant is simply $I_{R_2} = true$.

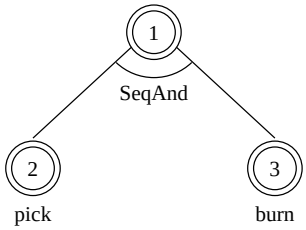

Figure 2: R_2's GDT

Actions

- $\texttt{pick}_{\texttt{R}_2}$:
  - preconditions: $\neg busy_{R_2} \wedge G(x_{R_2}, y_{R_2})$,
  - postconditions: $busy_{R_2} \wedge \neg G(x_{R_2}, y_{R_2})$.
- `burn`:
  - preconditions: $busy_{R_2}$,
  - postconditions: $\neg busy_{R_2}$.

GDT R_2's GDT is given on Figure 2. It has the following properties:

- Trigerring context: $TC_{R_2} = G(x_{R_2}, y_{R_2})$.
- Precondition: $PrecGDT_{R_2} = (\neg busy_{R_2})$.
- Initialization $\texttt{init}_{\texttt{R}_2}$: $busy_{R_2} = false$;.

Obviously, $\texttt{init}_{\texttt{R}_2}$ establishes $PrecGDT_{R_2}$ and I_{R_2}.

Obviously too, $PrecGDT_{R_2}$ is maintained by R_2's GDT, since the satisfaction condition of its top goal entails $\neg busy'_{R_2}$. This means that R_2 executes its GDT each time its trigerring context becomes true (i.e., each time a piece of garbage is on its cell).

4.1 Clean cell

Clean the cell.

Context: $\neg busy_{R_2} \wedge G(x_{R_2}, y_{R_2})$ (from $PrecGDT_{R_2}$ and TC_{R_2})
S.C.: $\neg busy_{R_2} \wedge \neg G(x_{R_2}, y_{R_2})$
Type: Non lazy, NS
Operator: SyncSeqAnd($G(x_{R_2}, y_{R_2})$)
Child(ren):

- Pick (4.2, p. 22)
- Burn (4.3, p. 22)

Proof: Obvious since $SC_{4.3} = SC_{4.1}$.

4.2 Pick (leaf)

Pick the piece of garbage on the cell.

Context: $\neg busy_{R_2} \wedge G(x_{R_2}, y_{R_2})$ (from $C_{4.1}$)
S.C.: $busy_{R_2} \wedge \neg G(x_{R_2}, y_{R_2})$
Type: Non lazy, NS
Action: $\texttt{pick}_{\texttt{R}_2}$

Proof: Obvious since the context entails the preconditions of action $\texttt{pick}_{\texttt{R}_2}$, and the postconditions of $\texttt{pick}_{\texttt{R}_2}$ entail $SC_{4.2}$.

Invariant: Obvious since $I_{R_2} = true$.

4.3 Burn (leaf)

Burn the piece of garbage currently held.

Context: $busy_{R_2} \wedge \neg G(x_{R_2}, y_{R_2})$ (from $SC_{4.2}$ since $G(x_{R_2}, y_{R_2})$ is synchronized in 4.1)
S.C.: $\neg busy_{R_2} \wedge \neg G(x_{R_2}, y_{R_2})$
Type: Non lazy, NS
Action: `burn`

Proof: Obvious since the precondition of the action is verified ($C_{4.3}$ entails $busy_{R_2}$), $\neg busy_{R_2}$ is proved by the postcondition of the action, and $G(x_{R_2}, y_{R_2})$ is not modified by the action.

Invariant: Obvious since $I_{R_2} = true$.