Appendix

1 Ontology
We present a semantic model of the building collar (UML class diagram) and an ontology of stairs (UML class diagram and OCL constraints).

1.1 UML
UML class diagram of entrances as a part of the building collar
UML class diagram of stairs.

1.2 OCL

1.2.1 Riser
context Riser inv:
  --thresholds for rise
  self.depth >= 14
  and
  self.depth <= 20

1.2.2 Tread
context Tread inv:
  --are horizontal
  self.normalVector.isParallel(0,0,1) = true
  and
  --thresholds for depth
  self.depth >= 18
  and
  self.depth <= 38
1.2.3  Step
context Step inv:

-- tread and riser are perpendicular
self.tread.normalVector.getAngle(self.riser.normalVector) = Pi/2
and

-- connection of tread and riser
-- for open stairs risers are non material
self.riser.upperCurve = self.tread.leadingCurve
and

-- pitch
self.pitch = self.riser.rise / self.tread.depth
-- azimuth
self.azimuth = self.riser.normalVector

1.2.4  Flight
context Flight inv:

-- same depth of all treads
self.step->forAll(s1, s2 : Step | s1.tread.depth = s2.tread.depth)
and

-- same height of all risers
self.step->forAll(s1, s2 : Step | s1.riser.rise = s2.riser.rise)
and

-- footstep formula 2r + t = 65
self.step.riser->forAll(r : Riser |
    self.step.tread->forAll(t : Tread |
        2 * r + t >= 53.6 and 2 * r + t <= 71.5
    )
)
and

-- each successor belongs to the same flight as his predecessor
self.step->includes(self.step.next)
and

-- connection of steps
self.step->forAll(s : Step | s.upperCurve = s.next.lowerCurve)
and

-- one exit step per flight
self.step->count(ExitStep) = 1
and

-- exit step is end of flight - no next
self.exitStep.next->isEmpty()
and

-- azimuth
self.step->forAll(s1, s2 : Step | s1.azimuth = s2.azimuth)

-- landings
context Landing inv:
self.normalVector.isParallel(0,0,1) = true

1.2.5  Staircase
context Staircase inv:

-- the number of flights at a landing
self.landing->forAll(l: Landing |
    l.connectionCurve->size() = l.lowerFlight->size() +
l.upperFlight->size()
)
and

-- connection of landing and flights
self.landing->forAll(l: Landing |
    l.connectionCurve->forAll(c : Curve |
        l.lowerFlight.upperCurve->includes(c)
or
        l.upperFlight.lowerCurve->includes(c)
    )
)
and

-- altitude difference = Staircase.numberOfSteps x Riser.rise
self.getAltitudeDifference() =
    sum(self.flight.riser.rise) * sum(self.flight.riser->size())
and
    --difference of the azimuth
    self.flight->forAll(f1, f2 : Flight |
        abs(f1.azimuth - f2.azimuth) = 0 or
        abs(f1.azimuth - f2.azimuth) = Pi/2 or
        abs(f1.azimuth - f2.azimuth) = Pi

1.2.6  Linear Staircase
context LinearStaircase inv:
    --two flights per landing
    self.landing->forAll(l : Landing | l.connectionCurve->size() = 2)
    and
    self.landing->forAll(l : Landing |
        --one upstairs flight
        l.lowerFlight->size() = 1
        and
        --and one downstairs flight
        l.upperFlight->size() = 1

1.2.7  Y-staircases
context YStaircase inv:
    --one landing
    self.landing->size() = 1
    and
    --with one upstairs flight and two downstairs flight (or vice versa)
    self.landing.connectionCurve->size() = 3
2 Grammar

2.1 Sets and Productions

2.1.1 Terminal symbols
- riser
- tread
- landing
- ( 
- )
- ;

2.1.2 Non-terminal symbols
- STAIR
- STAIRCASE
- LINEARSTAIRCASE
- Y-STAIRCASE
- FLIGHT
- STEP

2.1.3 Start symbol
STAIR

2.1.4 Productions
- STAIR ->
  o STAIRCASE
  o LINEARSTAIRCASE
  o Y-STAIRCASE
  o FLIGHT
- STAIRCASE ->
  o (STAIRCASE) landing (STAIRCASE)
  o STAIRCASE ; STAIRCASE
  o FLIGHT
- Y-STAIRCASE ->
  o FLIGHT landing (FLIGHT ; FLIGHT)
  o (FLIGHT ; FLIGHT) landing FLIGHT
- FLIGHT ->
  o STEP FLIGHT
  o STEP
- STEP -> riser tread

2.2 Attributes

2.2.1 riser
- rise : double
- width : double
- referencePoint : Point
- normalVector : Vector

2.2.2 tread
- depth : double
- width : double
- referencePoint : Point
- normalVector : Vector

2.2.3 landing
- width : double
- depth : double
- referencePoint : Point
- normalVector : Vector
2.2.4 STAIR
• numberOfSteps : int
• treadDepth : double
• riserRise : double
• pitch : Vector
• referencePoint : Point

2.2.5 STAIRCASE
• numberOfSteps : int
• treadDepth : double
• riserRise : double
• pitch : Vector
• referencePoint : Point

2.2.6 LINEARSTAIRCASE
• treadDepth : double
• riserRise : double
• pitch : Vector
• referencePoint : Point

2.2.7 Y-STAIRCASE
• treadDepth : double
• riserRise : double
• pitch : Vector
• referencePoint : Point

2.2.8 FLIGHT
• numberOfSteps : int
• treadDepth : double
• riserRise : double
• pitch : Vector
• referencePoint : Point
• azimuth : Vector

2.2.9 STEP
• rise : double
• depth : double
• pitch : double
• width : double
• referencePoint : Point
• azimuth : Vector

2.3 Semantic rules and guards
2.3.1 STAIR -> STAIRCASE
• STAIR.numberOfSteps = STAIRCASE.numberOfSteps
• STAIR.treadDepth = STAIRCASE.treadDepth
• STAIR.riserRise = STAIRCASE.riserRise
• STAIR.pitch = STAIRCASE.pitch
• STAIR.referencePoint = STAIRCASE.referencePoint

2.3.2 STAIR -> LINEARSTAIRCASE
• STAIR.numberOfSteps = LINEARSTAIRCASE.numberOfSteps
• STAIR.treadDepth = LINEARSTAIRCASE.treadDepth
• STAIR.riserRise = LINEARSTAIRCASE.riserRise
• STAIR.pitch = LINEARSTAIRCASE.pitch
• STAIR.referencePoint = LINEARSTAIRCASE.referencePoint
2.3.3  **STAIR -> Y-STAIRCASE**
- STAIR.numberOfSteps = Y-STAIRCASE.numberOfSteps
- STAIR.treadDepth = Y-STAIRCASE.treadDepth
- STAIR.riserRise = Y-STAIRCASE.riserRise
- STAIR.pitch = Y-STAIRCASE.pitch
- STAIR.referencePoint = Y-STAIRCASE.referencePoint

2.3.4  **STAIR -> FLIGHT**
- STAIR.numberOfSteps = FLIGHT.numberOfSteps
- STAIR.treadDepth = FLIGHT.treadDepth
- STAIR.riserRise = FLIGHT.riserRise
- STAIR.pitch = FLIGHT.pitch
- STAIR.referencePoint = FLIGHT.referencePoint

2.3.5  **STAIRCASE -> (STAIRCASE') landing (STAIRCASE'')**
- STAIRCASE.numberOfSteps = STAIRCASE'.numberOfSteps + STAIRCASE''.numberOfSteps
- STAIRCASE.treadDepth = weightedMean(STAIRCASE'.treadDepth, STAIRCASE''.treadDepth)
- STAIRCASE.riserRise = weightedMean(STAIRCASE'.riserRise, STAIRCASE''.riserRise)
- STAIRCASE.pitch = weightedMean(STAIRCASE'.pitch, STAIRCASE''.pitch)
- STAIRCASE.referencePoint = STAIRCASE'.referencePoint

2.3.6  **STAIRCASE -> STAIRCASE' ; STAIRCASE''**
- STAIRCASE.numberOfSteps = STAIRCASE'.numberOfSteps + STAIRCASE''.numberOfSteps
- STAIRCASE.treadDepth = weightedMean(STAIRCASE'.treadDepth = STAIRCASE''.treadDepth)
- STAIRCASE.riserRise = weightedMean(STAIRCASE'.riserRise = STAIRCASE''.riserRise)
- STAIRCASE.pitch = weightedMean(STAIRCASE'.pitch, STAIRCASE''.pitch)
- STAIRCASE.referencePoint = weightedMean(STAIRCASE'.referencePoint, STAIRCASE''.referencePoint)

2.3.7  **STAIRCASE -> FLIGHT**
- STAIRCASE.numberOfSteps = FLIGHT.numberOfSteps
- STAIRCASE.treadDepth = FLIGHT.treadDepth
- STAIRCASE.riserRise = FLIGHT.riserRise
- STAIRCASE.pitch = FLIGHT.pitch
- STAIRCASE.referencePoint = FLIGHT.referencePoint

2.3.8  **LINEARSTAIRCASE -> FLIGHT' landing FLIGHT''**
- LINEARSTAIRCASE.numberOfSteps = FLIGHT'.numberOfSteps + FLIGHT''.numberOfSteps
- LINEARSTAIRCASE.treadDepth = weightedMean(FLIGHT'.treadDepth, FLIGHT''.treadDepth)
- LINEARSTAIRCASE.riserRise = weightedMean(FLIGHT'.riserRise, FLIGHT''.riserRise)
- LINEARSTAIRCASE.pitch = weightedMean(FLIGHT'.pitch, FLIGHT''.pitch)
- LINEARSTAIRCASE.referencePoint = FLIGHT'.referencePoint

2.3.9  **Y-STAIRCASE -> FLIGHT' landing (FLIGHT'' ; FLIGHT''')**
- Y-STAIRCASE.numberOfSteps = FLIGHT'.numberOfSteps + FLIGHT''.numberOfSteps + FLIGHT''''.numberOfSteps
- Y-STAIRCASE.treadDepth = weightedMean(FLIGHT'.treadDepth, FLIGHT''.treadDepth, FLIGHT''''.treadDepth)
• Y-STAIRCASE.riserRise = weightedMean(FLIGHT'.riserRise, FLIGHT''.riserRise, FLIGHT'''.riserRise)
• Y-STAIRCASE.pitch = weightedMean(FLIGHT'.pitch, FLIGHT''.pitch, FLIGHT'''.pitch)
• Y-STAIRCASE.referencePoint = FLIGHT'.referencePoint

2.3.10 Y-STAIRCASE -> (FLIGHT'; FLIGHT'') landing FLIGHT'''
• Y-STAIRCASE.numberOfSteps = FLIGHT'.numberOfSteps + FLIGHT''.numberOfSteps + FLIGHT'''.numberOfSteps
• Y-STAIRCASE.treadDepth = weightedMean(FLIGHT'.treadDepth, FLIGHT''.treadDepth, FLIGHT'''.treadDepth)
• Y-STAIRCASE.riserRise = weightedMean(FLIGHT'.riserRise, FLIGHT''.riserRise, FLIGHT'''.riserRise)
• Y-STAIRCASE.pitch = weightedMean(FLIGHT'.pitch, FLIGHT''.pitch, FLIGHT'''.pitch)
• Y-STAIRCASE.referencePoint = weightedMean(FLIGHT'.referencePoint, FLIGHT''.referencePoint)

2.3.11 FLIGHT -> STEP FLIGHT'
• FLIGHT.numberOfSteps = FLIGHT'.numberOfSteps + 1
• FLIGHT.treadDepth = weightedMean(STEP.treadDepth, FLIGHT'.depth)
• FLIGHT.riserRise = weightedMean(STEP.riserRise, FLIGHT'.rise)
• FLIGHT.pitch = weightedMean(STEP.pitch, FLIGHT'.pitch)
• FLIGHT.pitch = STEP.pitch
• FLIGHT.referencePoint = STEP.referencePoint
• FLIGHT.azimuth = weightedMean(FLIGHT'.azimuth, STEP.azimuth)

2.3.12 FLIGHT -> STEP
• FLIGHT.numberOfSteps = 1
• FLIGHT.treadDepth = STEP.treadDepth
• FLIGHT.riserRise = STEP.riserRise
• FLIGHT.pitch = STEP.pitch
• FLIGHT.referencePoint = STEP.referencePoint
• FLIGHT.azimuth = weightedMean(STEP.azimuth)

2.3.13 STEP -> riser tread
• STEP.rise = riser.rise
• STEP.depth = tread.depth
• STEP.pitch = riser.rise / tread.depth
• STEP.width = weightedMean(riser.width, tread.width)
• STEP.referencePoint = riser.referencePoint
• STEP.azimuth = weightedMean(riser.normalVector)
3 ECLiPSe constraint solver program

:- lib(ic).

%%% observations
%%% values for observations, thresholds and accuracies
%%% observations
hp(p1, 22, 0).
hp(p2, 140, 75).
vp(p3, 71, 26).
hp(p4, 291, 115).

%%% thresholds for treads
tread_min(18).
tread_max(38).

%%% thresholds for risers
riser_min(14).
riser_max(20).

%%% accuracies
delta_x_horizontal(1).
delta_x_vertical(1).
delta_y_horizontal(1).
delta_y_vertical(1).

%%% constraint rules
%%% rules for estimating the reference points from points laying
%%% on horizontal or vertical faces
%%% reference point: connecting point for risers and treads
%%% estimate_RefX_from_hPoint(P_i, RefX, Tread, N, X0) :-
hp(P_i, X, _),
delta_x_horizontal(Delta_x_horizontal),
X $\geq$ RefX - Delta_x_horizontal,
X $\leq$ RefX + Delta_x_horizontal + Tread,
RefX $=$ X0 + N * Tread.

estimate_RefY_from_hPoint(P_i, RefY, Riser, M, Y0) :-
hp(P_i, _, Y),
delta_y_vertical(Delta_y_vertical),
Y $\geq$ RefY - Delta_y_vertical,
Y $\leq$ RefY + Delta_y_vertical,
RefY $=$ Y0 + M * Riser.

estimate_RefX_from_vPoint(P_i, RefX, Tread, N, X0) :-
vP(P_i, X, _),
delta_y_horizontal(Delta_y_horizontal),
X $\geq$ RefX - Delta_y_horizontal,
X $\leq$ RefX + Delta_y_horizontal,
RefX $=$ X0 + N * Tread.

estimate_RefY_from_vPoint(P_i, RefY, Riser, M, Y0) :-
vP(P_i, _, Y),
delta_y_vertical(Delta_y_vertical),
Y $\geq$ RefY - Delta_y_vertical,
Y $\leq$ RefY + Delta_y_vertical + Riser,
RefY $=$ Y0 + M * Riser.
treppe(Tread, Riser, X0, Y0, X1, Y1, N1, M1, X2, Y2, N2, M2, X3, Y3, N3, M3, X4, Y4, N4, M4) :-

% reading of values defined above
NMs = [N1, N2, N3, N4, M1, M2, M3, M4],
NMs #:: 0 .. 18,
Xs = [X0, X1, X2, X3, X4],
Xs $:: 0 .. 684,
Ys = [Y1, Y2, Y3, Y4],
Ys S:: 0 .. 360,

% thresholds for treads
Tread $=< Tread_Max,
Tread $>= Tread_Min,

% thresholds for risers
Riser $=< Riser_Max,
Riser $>= Riser_Min,

% accuracy
delta_x_horizontal(Delta_x_horizontal),
delta_y_vertical(Delta_y_vertical),

% defining constraints

%estimating X values of reference points
estimate_RefX_from_hPoint(p1, X1, Tread, N1, X0),
estimate_RefX_from_hPoint(p2, X2, Tread, N2, X0),
estimate_RefX_from_vPoint(p3, X3, Tread, N3, X0),
estimate_RefX_from_hPoint(p4, X4, Tread, N4, X0),

%estimating Y values of reference points
estimate_RefY_from_hPoint(p1, Y1, Riser, M1, Y0),
estimate_RefY_from_hPoint(p2, Y2, Riser, M2, Y0),
estimate_RefY_from_vPoint(p3, Y3, Riser, M3, Y0),
estimate_RefY_from_hPoint(p4, Y4, Riser, M4, Y0),

labeling([N1, N2, N3, N4, M1, M2, M3, M4]).

% short form
%call: t([Tread, Riser, Tread_median, Rise_median, X0, Y0, N1, M1, N2, M2, N3, M3, N4, M4])
\[ t([\text{Tread, Riser, Tmed, Rmed, X0, Y0, N1, M1, N2, M2, N3, M3, N4, M4}]):- \]

\[
\text{treppe(} \\text{Tread, Riser,} \\
X0, Y0, \\text{X1, Y1, N1, M1,} \\
X2, Y2, N2, M2, \\text{X3, Y3, N3, M3,} \\
\text{X4, Y4, N4, M4),} \]

\[
\text{get\_median(Tread, Tmed),} \\
\text{get\_median(Riser, Rmed).} \]

4 Database

Distribution of the main stair parameters rise and tread.

5 Images
The figure shows three different types of flights: steps without nosing (left), steps with nosing (middle) and open steps (right) i.e. no risers between treads.