Reasoning in LoCo with Answer Set Programming

M.Aschinger, C. Drescher, G. Gottlob

2012 Oxford Configuration Workshop

13.01.2012

Transformation to target languages

- LoCo is a high-level representation language
- declarative modelling without knowledge about underlying solving algorithms
- automated transformation into target languages such as
  - ASP (current work)
  - SAT, Integer Programming, Constraint Solving
  - High-level languages for combinatorial optimisation, e.g. MiniZinc
- restriction to finite models: finite upper bounds on all components via Bounds Propagation algorithm

Necessary steps

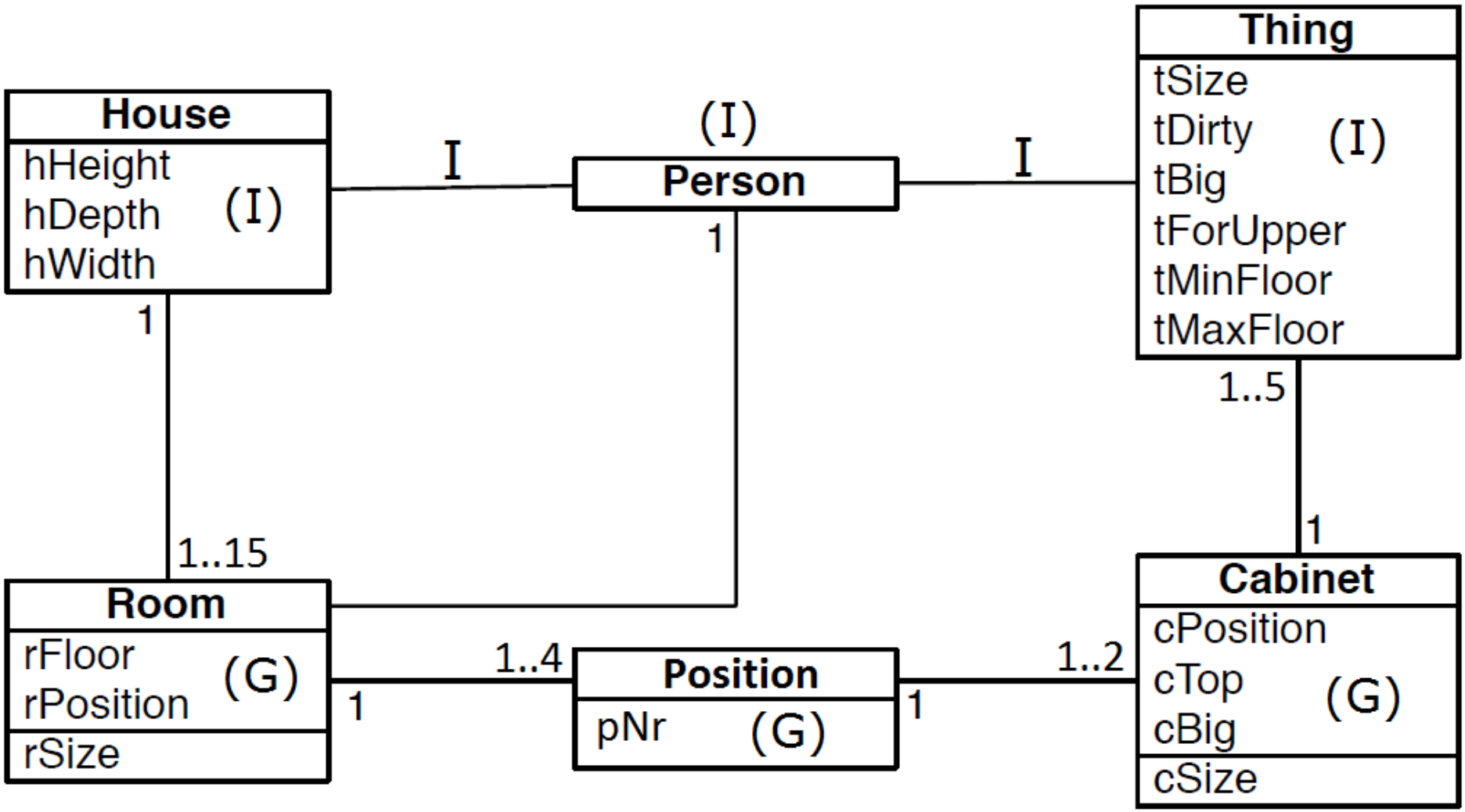
- Problem analysis
- Definition of LoCo Domain Knowledge
- Putting problem instances into LoCo Instance Knowledge format
- Computation of bounds
- Transformation into target language

Problem description of the House Problem

House Problem

- toy version of a real-world problem: placement of modules in racks with lots of complicated constraints
- basically sort of a modified Bin-Packing problem
- **TASK:** A house is inhabited by people owning things of various types and sizes. Those things have to be stored in cabinets in rooms of the house. The type, size and other attributes put constraints on to where a thing can be stored.
- **GOAL:** Find a minimal number of cabinets, counting twice all big cabinets.

House Problem



LoCo Domain Knowledge

Constants and components

- Constants

  const cabMaxSize.
  const roomCabSpace.
  const maxHouseHeight.
  const maxHouseWidth.

- Components

  house: IC (ID; height:Integer; width:Integer)
  person: IC (ID)
  thing: IC (ID; size:Integer[0..cabMaxSize]; big:Bool; dirty:Bool; forUpper:Bool)
  cabinet: GC (ID; size:Integer[0..cabMaxSize]; big:Bool; dirty:Bool; top:Bool)
  position: GC (ID; number:Integer[1..roomCabSpace])
  room: GC (ID; floor:Integer[0..maxHouseHeight]; pos:Integer[0, maxHouseWidth])

Connection predicates

- Binary connections

  house2Person: IIC (house x person)
  person2Thing: IIC (person x thing)
  thing2Cabinet: IGC (thing x cabinet)
  cabinet2Position: GGC (cabinet x position)
  position2Room: GGC (position x room)
  room2Person: GIC (room x person)

Connection axioms

Thing $\rightarrow$ Cabinet:

$$(\forall t)\mathit{thing}(t) \Rightarrow (\exists_1^1 c)\mathit{cabinet}(c) \wedge \mathit{thing2Cabinet}(t,c) \wedge [(c.\mathit{big} \wedge t.\mathit{big}) \vee \neg t.\mathit{big}]$$

Cabinet $\rightarrow$ Thing:

$$(\forall c)\mathit{cabinet}(c) \Rightarrow (\exists_1^{\mathit{cabMaxSize}} t)\mathit{thing}(t) \wedge \mathit{thing2Cabinet}(t,c) \wedge (\sum t.\mathit{size} < \mathit{cabMaxSize}) \wedge (t.\mathit{dirty} == c.\mathit{dirty}) \wedge (\neg c.\mathit{top} \vee t.\mathit{forUpper})$$

Non-local constraints

Integrity rules:

$$(\forall c)\mathit{cabinet}(c) \wedge c.\mathit{big} \Rightarrow \neg c.\mathit{top}$$

$$(\forall r1, r2)\mathit{room}(r1) \wedge \mathit{room}(r2) \wedge r1.\mathit{floor} = r2.\mathit{floor} \wedge$$
$$r1.\mathit{position} = r2.\mathit{position} \Rightarrow r1 = r2$$

Connection-generating rules:

$$(\forall\ \mathit{pe}, t, c, \mathit{po}, r)\mathit{person2Thing}(\mathit{pe}, t) \wedge \mathit{thing2Cabinet}(t, c) \wedge$$
$$\mathit{cabinet2Position}(c, \mathit{po}) \wedge \mathit{position2Room}(\mathit{po}, r) \Rightarrow$$
$$\mathit{room2Person}(r, \mathit{pe})$$

LoCo Instance Knowledge and Bounds Computation

LoCo Input file

- Constants

  const cabMaxSize = 5.
  const roomCabSpace = 4.
  const maxHouseHeight = 10.
  const maxHouseWidth = 20.

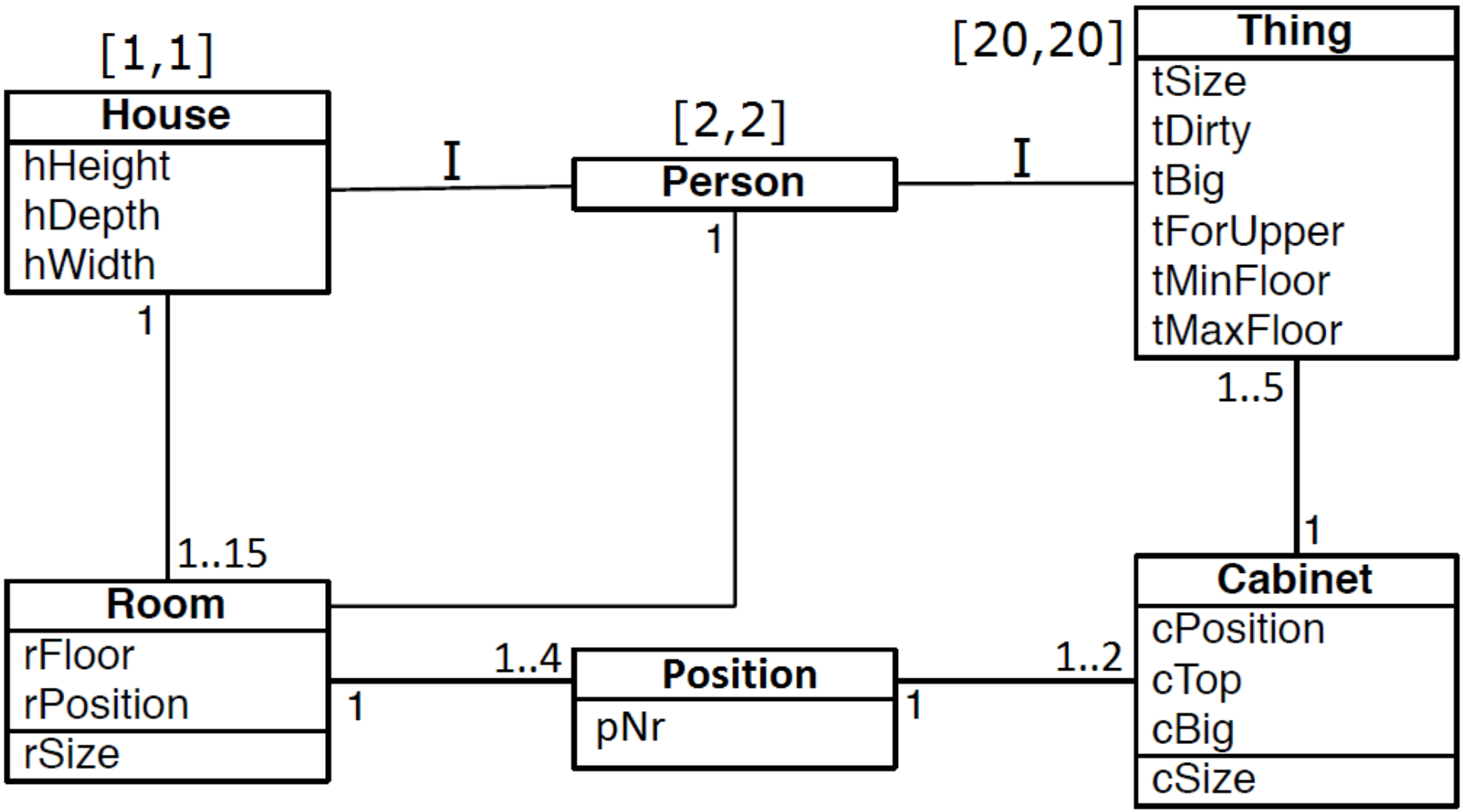
- Input components

  house(1).
  person(1..2).
  thing(1..20).
  thingSize(1,4). thingBig(1,0). thingDirty(1,1). ...
  ...

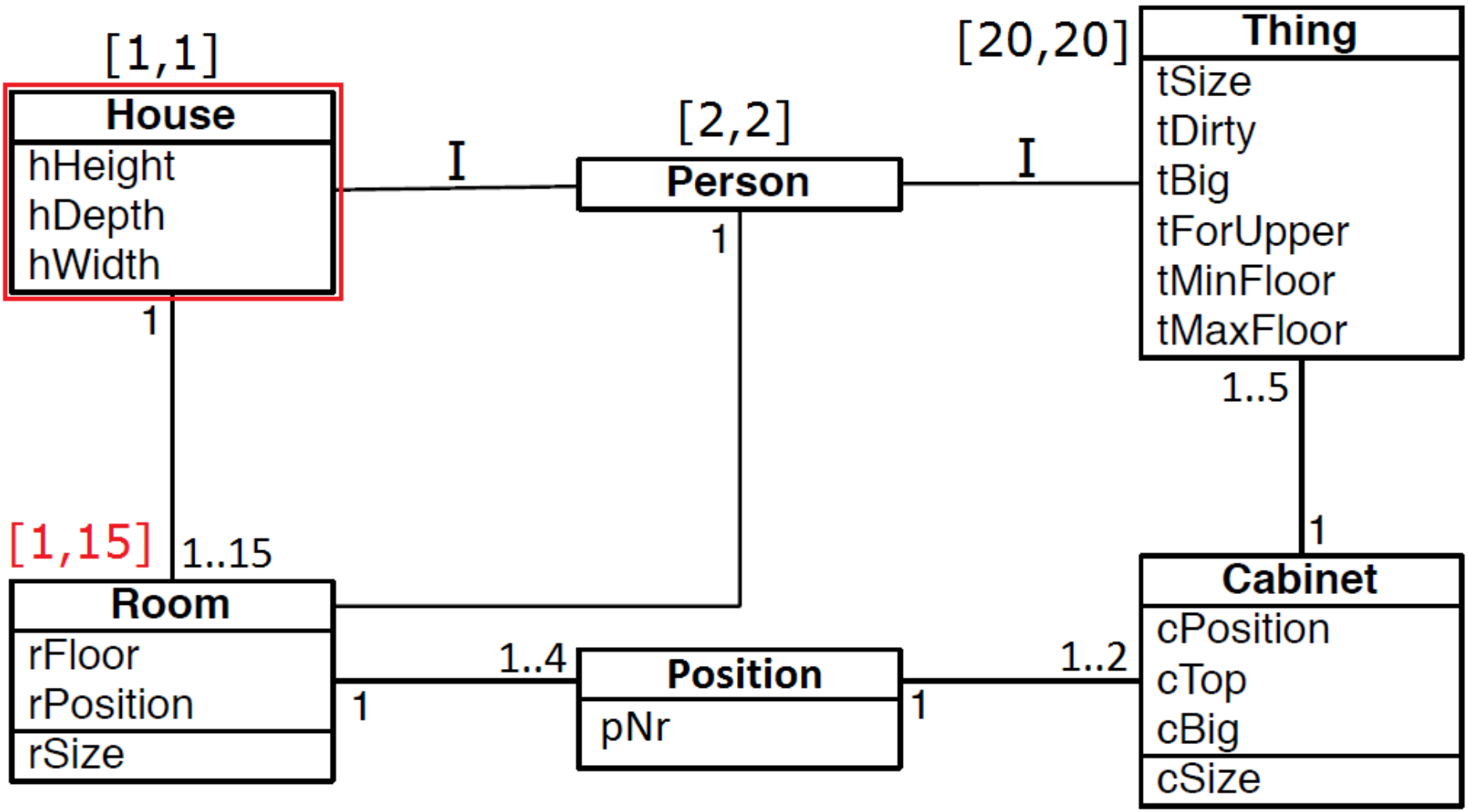
- Input connections

  house2Person(1,1). house2Person(1,2).
  person2Thing(1,3). person2Thing(1,1).
  person2Thing(2,2). person2Thing(2,4). ...
  ...

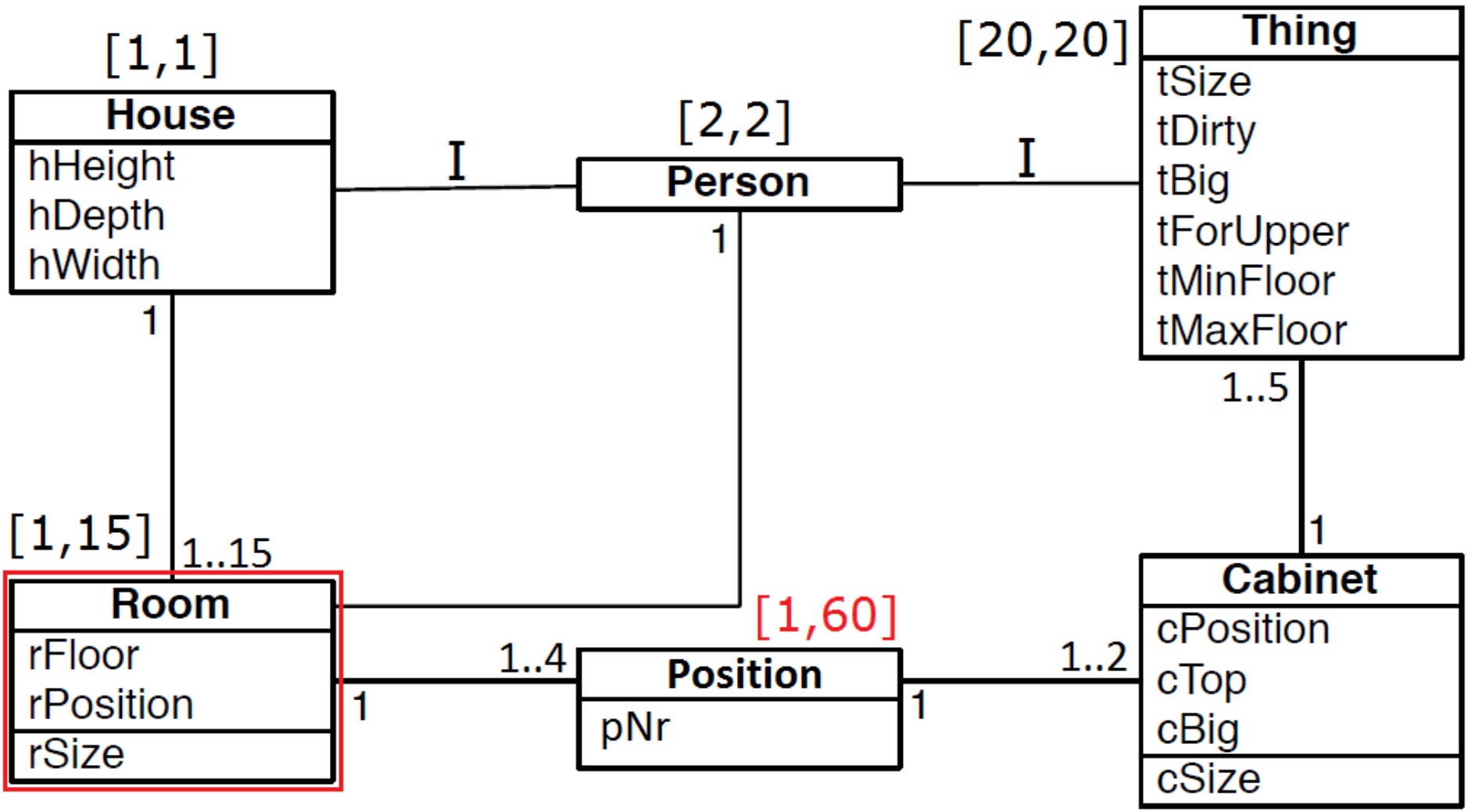
Bounds Computation



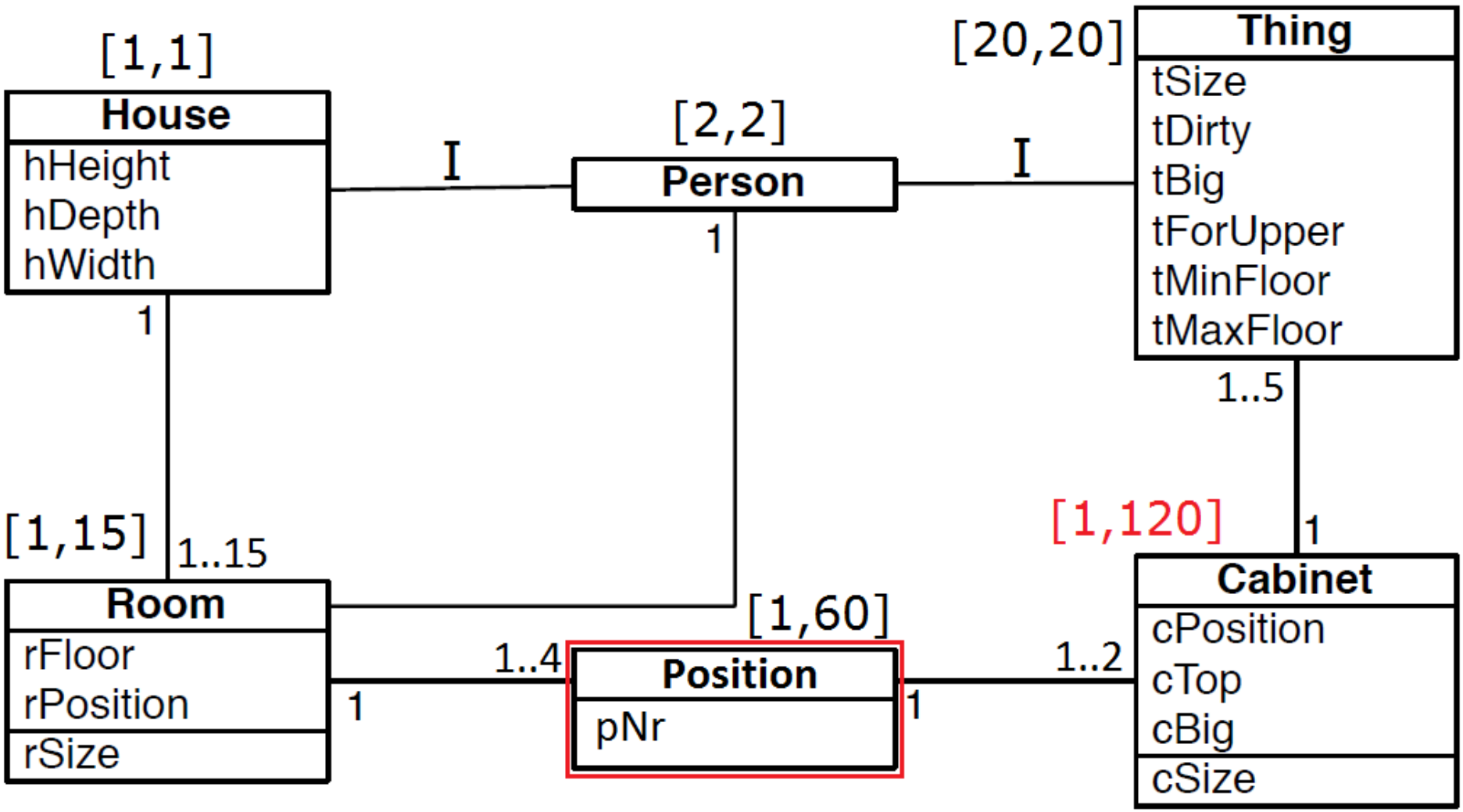
Bounds Computation



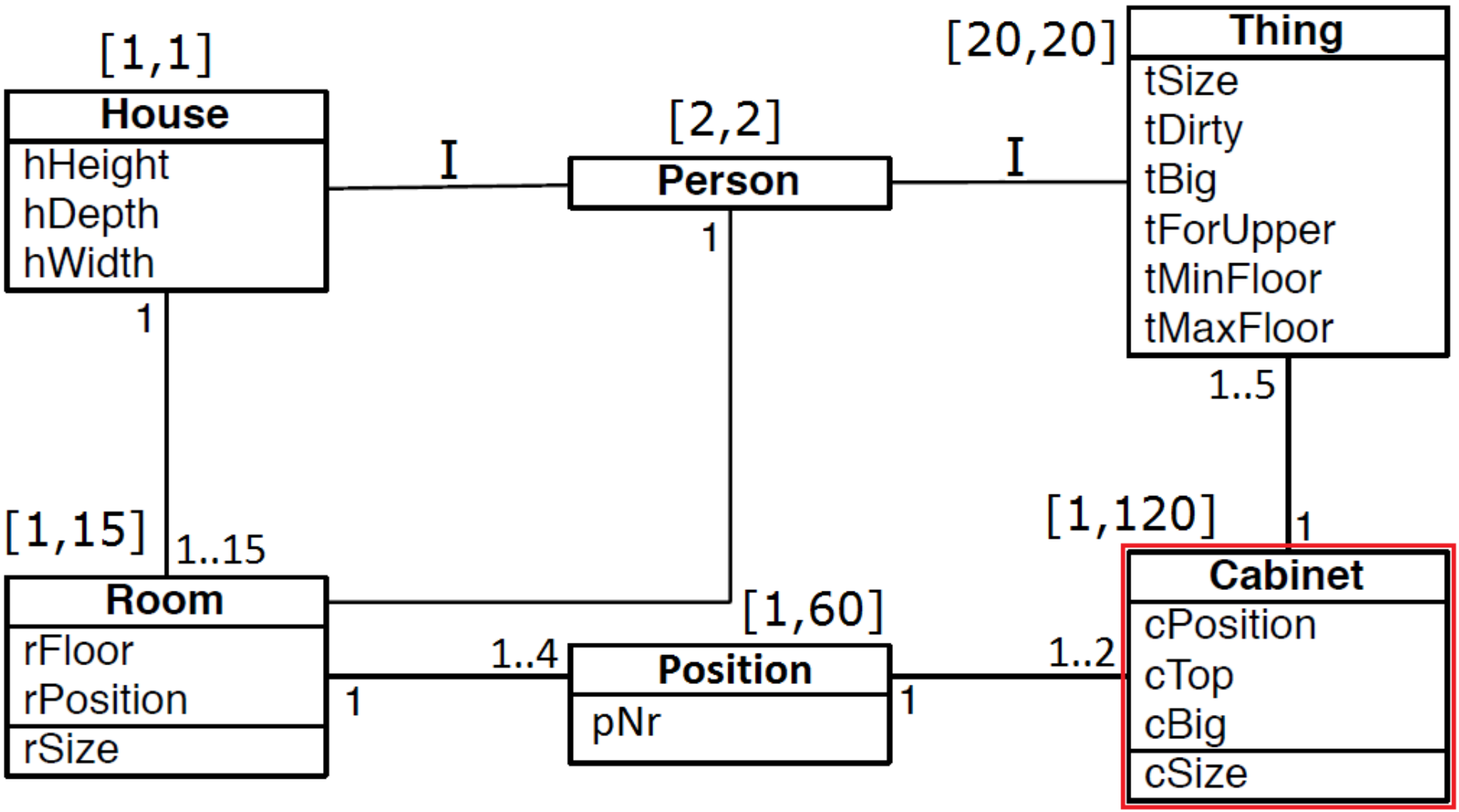
Bounds Computation



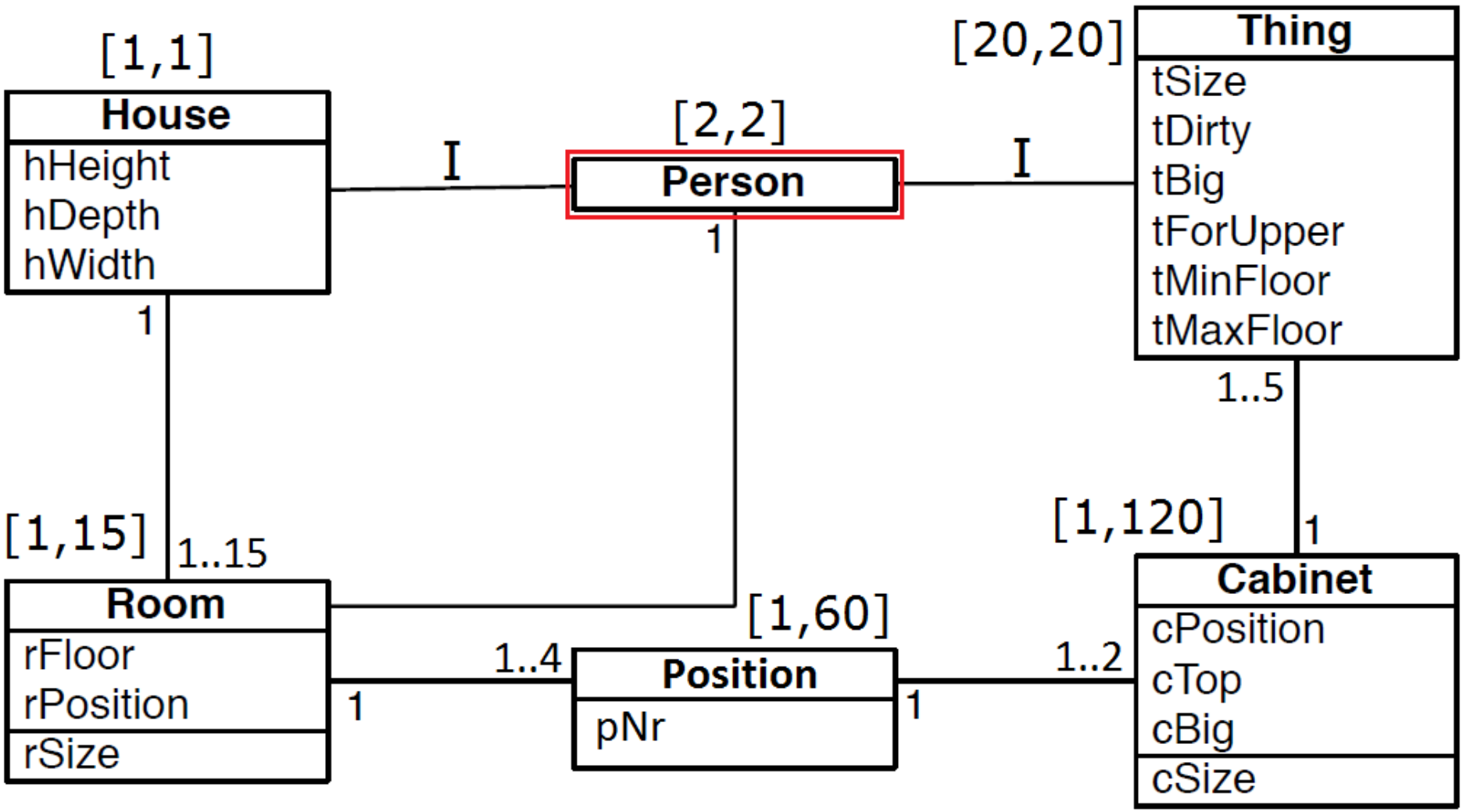
Bounds Computation



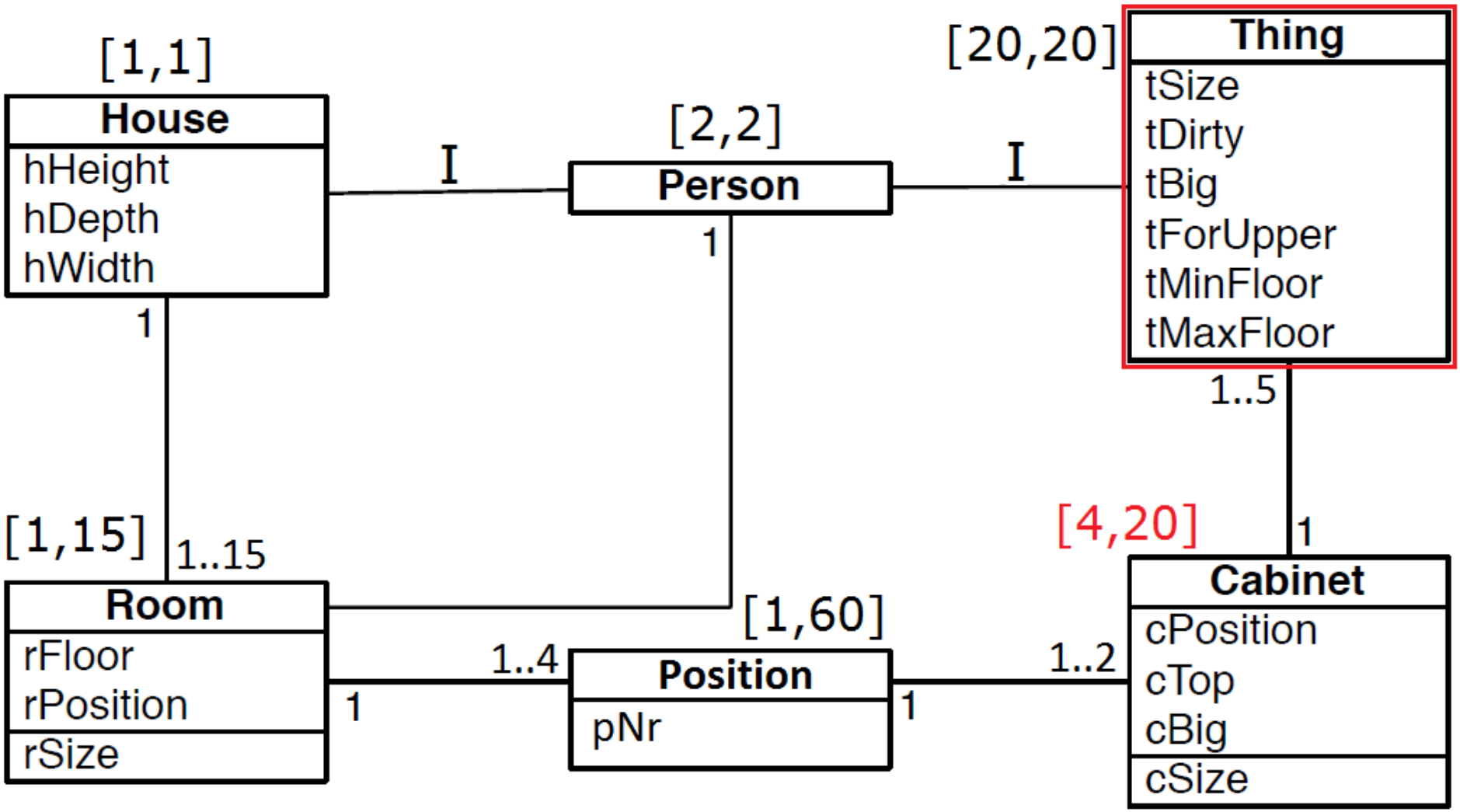
Bounds Computation



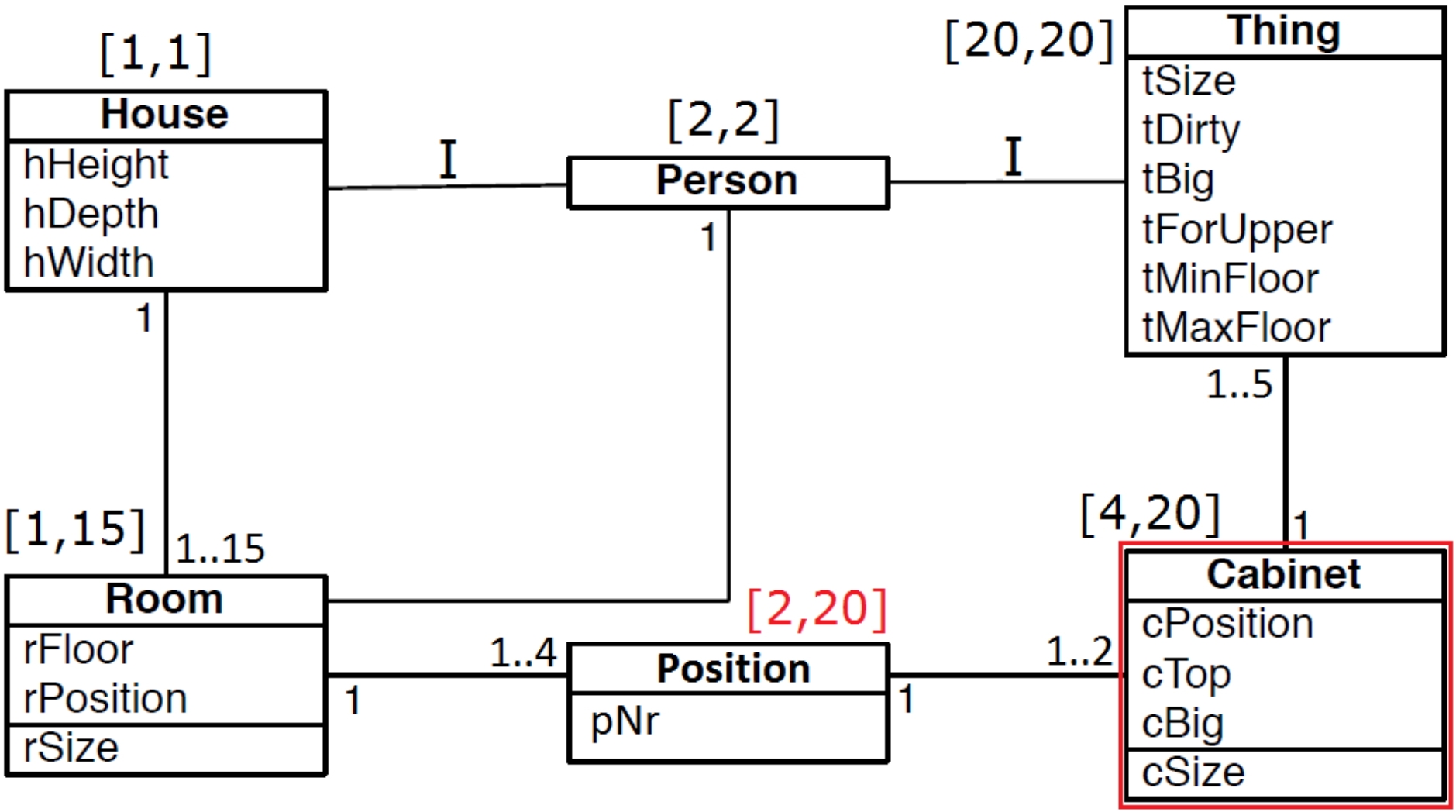
Bounds Computation



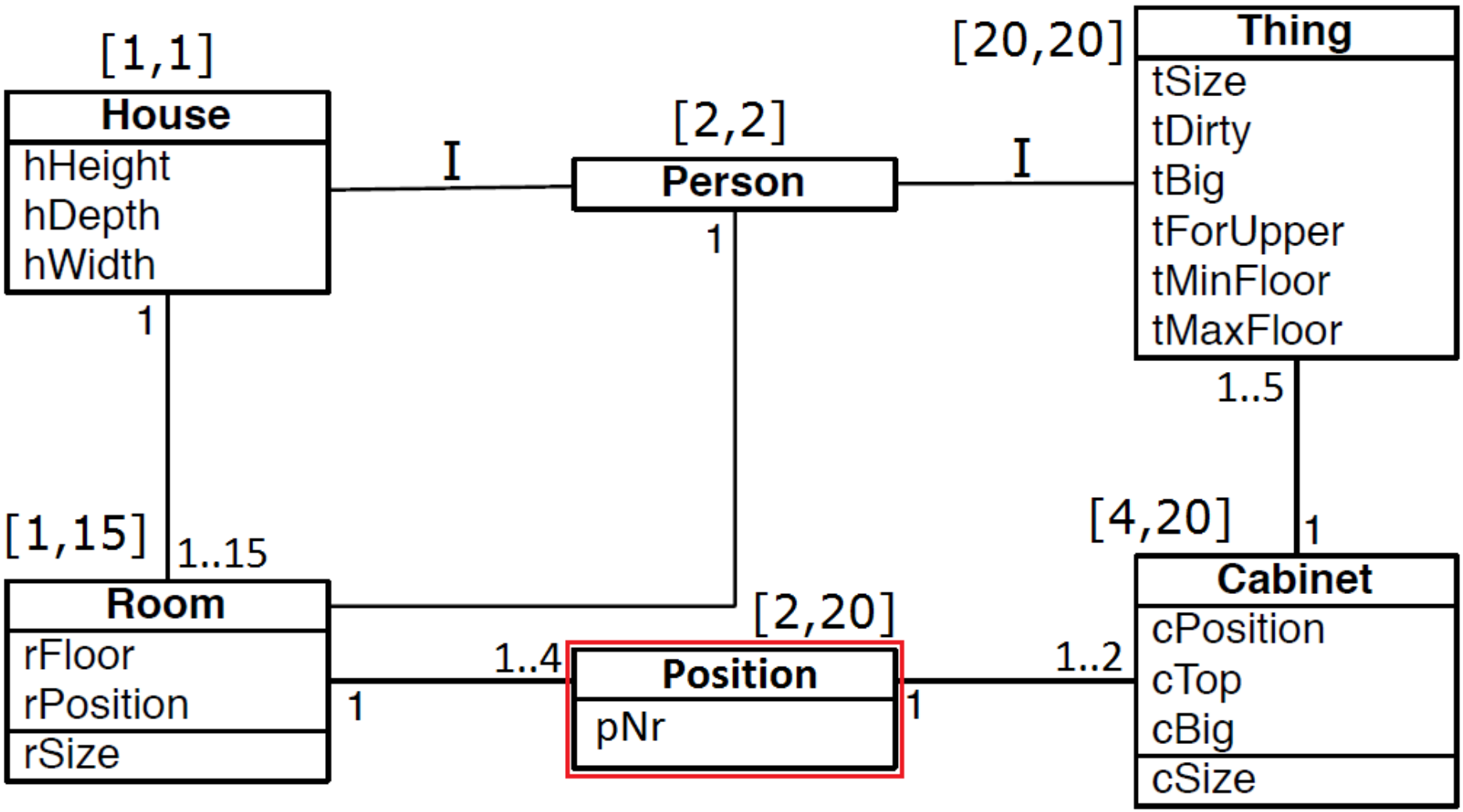
Bounds Computation



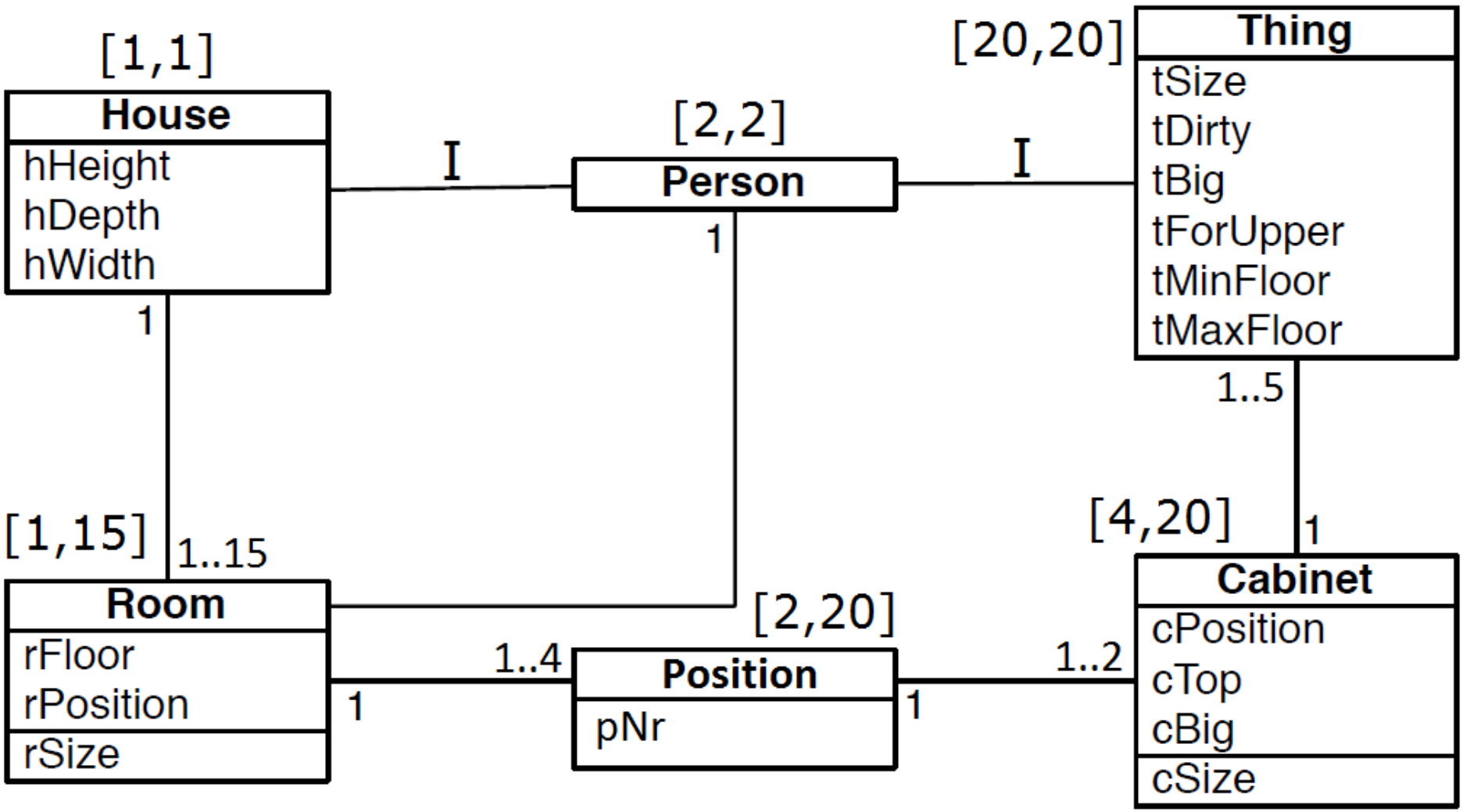
Bounds Computation



Bounds Computation



Bounds Computation



Digression: Zeros and one-to-many

$$(\forall s)sensor(s) \Rightarrow (\exists_1^1\ u)unit(u) \wedge sensor2Unit(s,u)$$
$$(\forall u)unit(u) \Rightarrow (\exists_0^2\ s)sensor(s) \wedge sensor2Unit(s,u)$$

$$(\forall z)zone(z) \Rightarrow (\exists_1^1\ u)unit(u) \wedge zone2Unit(z,u)$$
$$(\forall u)unit(u) \Rightarrow (\exists_0^2\ z)zone(z) \wedge zone2Unit(z,u)$$

$$(\forall u)unit(u) \Rightarrow (\exists_1^4\ x)$$
$$[(sensor(x) \wedge sensor2Unit(x,u)) \vee$$
$$(zone(x) \wedge zone2Unit(x,u))]$$

Transformation to ASP

Constants and components

- Constants

  straight-forward

- Components

  cabinetGen(1..20).
  4 { cabinet(C) : cabinetGen(C) } 20.

- Attributes

  1 { cabinetSize(C,S) : S = 0..cabMaxSize } 1 ← cabinet(C).
  1 { cabinetDirty(C,D) : D = 0..1 } 1 ← cabinet(C).
  1 { cabinetBig(C,B) : B = 0..1 } 1 ← cabinet(C).
  1 { cabinetTop(C,T) : T = 0..1 } 1 ← cabinet(C).

Binary connections

Thing $\rightarrow$ Cabinet:

$$(\forall t)\mathit{thing}(t) \Rightarrow (\exists_1^1 c)\mathit{cabinet}(c) \wedge \mathit{thing2Cabinet}(t,c) \wedge [(c.\mathit{big} \wedge t.\mathit{big}) \vee \neg t.\mathit{big}]$$

1 { thing2Cabinet(T,C) : cabinetGen(C) } 1 $\leftarrow$ thing(T).

1 { thing2Cabinet(T,C) : thing(T) } cabMaxSize $\leftarrow$ cabinet(C).

$\leftarrow$ thing2Cabinet(T,C), not cabinet(C).

Binary connections

Thing $\rightarrow$ Cabinet:

$$(\forall t)thing(t) \Rightarrow (\exists_1^1 c)cabinet(c) \wedge thing2Cabinet(t,c) \wedge [(c.big \wedge t.big) \vee \neg t.big]$$

$\leftarrow$ not C1(T,C), thing2Cabinet(T,C).

C1(T,C) $\leftarrow$ C1_1(T,C).

C1(T,C) $\leftarrow$ C1_2(T,C).

C1_1(T,C) $\leftarrow$ C1_11(T,C), C1_12(T,C).

C1_2(T,C) $\leftarrow$ thing2Cabinet(T,C), thingBig(T,TB), TB == 0.

C1_11(T,C) $\leftarrow$ thing2Cabinet(T,C), cabinetBig(C,CB), CB == 1.

C1_12(T,C) $\leftarrow$ thing2Cabinet(T,C), thingBig(C,CB), TB == 1.

Ongoing and Future Work

- Benchmarks
- Determine complexity of reasoning tasks (at least NP hard)
- Identify tractable islands
- Map to executable formats
- Graphical User Interface
- Extended optimization functionality

Thank you for your attention