Knowledge Representation and Reasoning - H02C3A

Exercise Session 3

Professor Marc Denecker (marc.denecker@cs.kuleuven.be)
Teaching assistant: Pierre Carbonnelle (pierre.carbonnelle@kuleuven.be)

1 FO(ID): semantics

Assignment Check if the following definition Δ can be stratified. (hint: $P \leftarrow$ is the same as $P \leftarrow true$.) If so, construct its model by iterated rule application along its dependency graph. Also, compute the justification tree for T relative to this definition. Do all defined atoms have a two-valued supported value?

Note: a *well-founded interpretation* of a definition is one such that the value of each defined atom is equal to its supported value. The well-founded interpretation is called a *model* of the definition if it is 2-valued. Does this definition have a model?

$$\Delta = \left\{ \begin{array}{l} P \leftarrow \\ Q \leftarrow \neg P \\ Q \leftarrow R \\ R \leftarrow Q \\ S \leftarrow \neg Q \\ T \leftarrow S \wedge \neg U \\ U \leftarrow \neg P \vee Q \end{array} \right\}$$

Solution Stratification of an inductive definition splits the definitional rules up over disjunct *layers* or *strata*. The definitional rules are split up in layers corresponding to the implicit induction order of the definition. This is useful if you want to apply iterated rule application to compute models of the inductive definition. To construct the sets corresponding to the defined predicates, you start by applying the rules in the first layer, until there are no longer rules from that layer that can be applied. Once there are no longer rules that can be applied, you move over to the second layer, and apply rules from the second layer until there are no longer rules that can be applied. In this way, you go once over over all layers in order, applying only rules from the current layer.

Note however that if no induced induction order exists, stratification is not possible. You should then try the method with justifications to compute a model. Also note that an inductive definition is stratified if it *can* be divided into strata. This is a sufficient, but not necessary condition (see the Lemma further below).

To see if an inductive definition is stratified, one can construct a graph based on the dependency relation of the definition, i.e. a *dependency graph*. The dependency relation $\triangleleft$ for a definition Δ is the least transitive relation $\triangleleft$ such that $B \triangleleft A$ if

- Δ contains a rule $A \leftarrow \phi$, and
- B occurs in ϕ

That is, A *depends on* B, or A *refers to* B.

If the inductive definition contains a clause of the form

$$p(\dots) \leftarrow \dots, q(\dots), \dots$$

then predicate p depends *positively* on q. The positive literal q has to be defined in *the same layer as* p*, or in an earlier layer.*

If the definition contains a clause of the form

$$p(\dots) \leftarrow \dots, \text{not } q(\dots), \dots$$

then predicate p depends *negatively* on q. The negative literal not $q(\dots)$ has to be defined in a ***strictly*** *earlier layer than* p.

The dependency graph of Δ is constructed by creating a node for each defined predicate. For each dependency, a directed edge is created, pointing towards the predicate that depends on the other predicate. The dependency edges are labeled with a $+$ or a $-$, indicating if the dependency is respectively positive or negative.

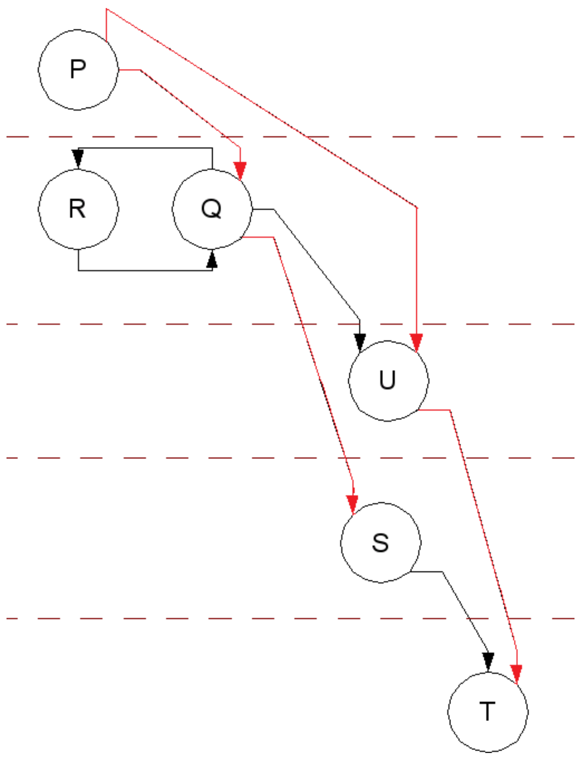

Figure 1: The dependency graph for Δ contains no cycles containing a negative edge. Δ can be stratified.

In this exercise, the definition contains defined predicates $def(\Delta) = \{P, Q, R, S, T, U\}$ and parameter symbols $pars(\Delta) = \emptyset$. The dependency graph for Δ definition is shown in figure 1, with the red edges indicating negative dependencies. There are three negative dependencies: Q depends negatively on P, S on Q, and T on U.

The following lemma can then be used to verify whether the definition is stratified.

Lemma. *An inductive definition Δ is stratified $\iff$ the dependency graph for Δ contains no cycles containing a negative edge.*

The dependency graph of the exercise definition Δ contains no cycles, so it is stratified.

One or more stratifications of the definition can be derived from the dependency graph (or directly from the dependencies). The following stratification into 3 layers is sufficient.

$$\left\{\begin{array}{l} P \leftarrow \\ ----- \\ Q \leftarrow \neg P \\ Q \leftarrow R \\ R \leftarrow Q \\ U \leftarrow \neg P \vee Q \\ ----- \\ S \leftarrow \neg Q \\ T \leftarrow S \wedge \neg U \end{array}\right\}$$

This is a correct stratification: each layer can safely be computed based on the interpretation obtained in previously-computed (lower-numbered) layers. *layer 2* contains two negative dependencies on P, which is defined in *layer 1*. *Layer 3* contains negative dependencies on Q and U, which are defined in *layer 2*. Given that the definition has a correct stratification, it has a two-valued interpretation and hence has a model.

After a correct stratification, a model for the definition can then be constructed using iterated rule application. One starts with a structure $\mathcal{O}$ for the parameter symbols $pars(\Delta)$. In this example, $def(\Delta) = \{P, Q, R, S, T, U\}$ and $pars(\Delta) = \emptyset$, which means $\mathcal{O} = \emptyset$. There are no parameter symbols, i.e., symbols without definitions.

We can compute the model layer by layer.

- Layer 1:
  - P is defined from true, and is thus true.
  - Hence, after layer 1, $\mathcal{I}_1 = \{P\}$.
- Layer 2:
  - We can safely conclude that first rule defining Q is not applicable in $\mathcal{I}_1$ because we know from the previous step that P is true and hence $\neg P$ is false.
  - For the other three rule, we apply iterated rule application. The second rule defining Q and the rule defining R form a positive loop ($Q \leftarrow R$ and $R \leftarrow Q$). None of these two rules can ever fire in $\mathcal{I}_1$, so both Q and R cannot be added to the model.
  - Similarly, we know that the rule defining U is not applicable in $\mathcal{I}_1$. Since this is the only rule defining U, U cannot be added to the model.
  - Thus, after layer 2, $\mathcal{I}_2 = \{P\}$.
- Layer 3:
  - We can now safely conclude that $\neg Q$ is true in $\mathcal{I}_2$ and hence S becomes true.
  - We also know that $\neg U$ is true, so T becomes true.
  - Hence, after layer 3, $\mathcal{I}_3 = \{P, S, T\}$

The model is then $M = \{P, S, T\}$ (i.e., P, S and T are true, the other propositions are false).

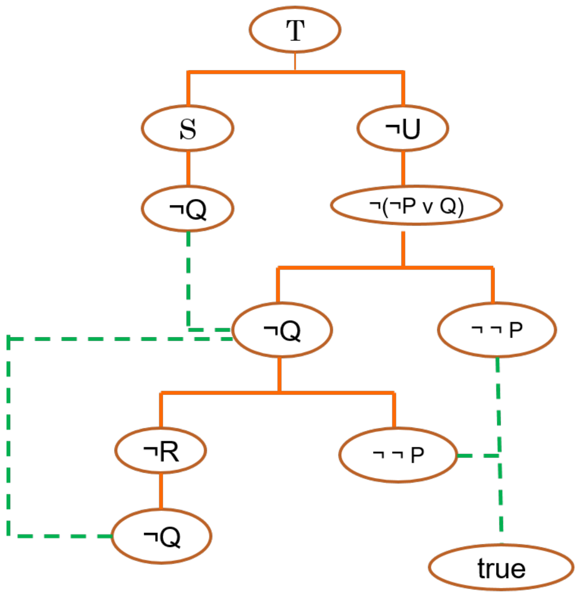

Figure 2: A justification for T relative to $Delta$.

Justification tree If iterated rule application is not possible, you can build a *justification (tree)*, starting from a symbol in the lowest part of the dependency graph. The justification tree for $T = t$ relative to Δ is shown in figure 2. Note that the notation is slightly different from the one in Lecture 4: $T = t$ is denoted T here, and $T = f$ is denoted $\neg T$ here. Each branching denotes a conjunction (”and”); several trees would be used to represent a disjunction, i.e. different possible justifications of the same fact $T = t$. The value of a justification for $T = t$ is determined by the weakest path in the strongest justification tree. Notice that one of the paths leads into a cycle of negative defined base literals $\neg Q$ and $\neg R$, so the value of that path is positive (i.e., $\neg Q$ and $\neg R$ are true). The finite paths ending at defined base literal P are true, since P is defined as true. For the formula $\neg Q$, the justification (sub)tree contains two paths: one is cycle of negative defined base literals (value true) and the other end in P. So $\neg Q$ is justified to be true. So all the paths for the justification of T are true, so the T is also justified to be true. To conclude, the justification tree contains true justifications for $T, S, \neg U, \neg Q, P$. All the defined literals are two-valued (i.e., they have a supported value equal to *true* or *false*, but never *undefined*). The model is $M = \{P, S, T\}$.

1.1 Auxiliary exercise (homework):

Assignment Build a justification tree for $Even(4)$ given the definition below and given domain of natural numbers $D = \{0, 1, 2, ..., \infty\}$ and having that $0^I = 0$ (in interpretation I constant 0 is mapped to zero) and $S^I = \{(0, 1), (1, 2), (2, 3), ..., \}$ (successor function).

$$\left\{ \begin{array}{c} Even(x) \leftarrow x = 0 \\ Even(S(x)) \leftarrow Odd(x) \\ Odd(S(x)) \leftarrow Even(x) \end{array} \right\}$$

Solution Figure 3 gives a justification for $Even(4)$.

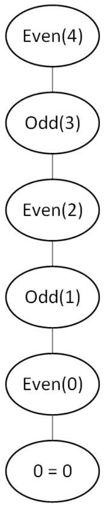

Figure 3: Justification tree for $Even(4)$.

Hence, 4 is even.

2 FO(ID): semantics, (in)consistent definitions

Assignment Given the following inductive definitions.

$$\Delta_1 = \begin{Bmatrix} P \leftarrow Q \\ Q \leftarrow P \end{Bmatrix}$$

$$\Delta_2 = \begin{Bmatrix} P \leftarrow Q \\ Q \leftarrow \neg P \end{Bmatrix}$$

$$\Delta_3 = \begin{Bmatrix} P \leftarrow \neg Q \\ Q \leftarrow \neg P \end{Bmatrix}$$

What are the well-founded interpretations of these definitions? Which definitions have a model? Which definitions can be stratified? (Also construct justification trees).

Solution Each of the three definitions has defined predicates $def(\Delta) = \{P, Q\}$ and parameter symbols $pars(\Delta) = \emptyset$. They each contain a cycle. The dependency graphs are in figure 4. Justifications for P are given in figure 5 (again, with a notation slightly different from the one in the lecture).

1. The first definition Δ_1 contains no negation. Its dependency graph is a *positive cycle*. This definition is also stratified, since it has no loops containing negative edges. However, because of the loop, the stratification contains only one layer.
   Δ_1 has a two-valued well-founded interpretation $\mathcal{I}_{\Delta_1} = \emptyset$. The values of P and Q in this interpretation are false, that is $P^{\mathcal{I}_{\Delta_1}} = Q^{\mathcal{I}_{\Delta_1}} = false$. This follows from the inductive construction process. We start with a structure $\mathcal{O} = \emptyset$ for the parameter symbols of Δ_1, for which $P^{\mathcal{O}} = Q^{\mathcal{O}} = false$.

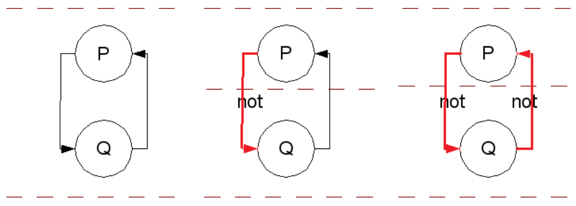

Figure 4: Dependency graph for the exercise on (in)consistent definitions.

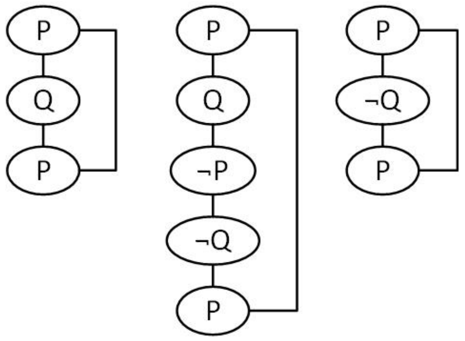

Figure 5: Justifications for P relative to Δ_1, Δ_2 and Δ_3, respectively.

Because none of the two rules can fire, P and Q cannot be added to $\mathcal{I}_{\Delta_1} = \mathcal{O}$. Since $\mathcal{I}_{\Delta_1}$ is a two-valued well-founded interpretation of Δ_1, it is a model of Δ_1.

2. The dependency graph of Δ_2 contains a negative edge, because Δ_2 contains the negative literal $\neg P$ in the body of a rule. Therefore, it cannot be stratified. The justification for P relative to Δ_2 is one infinite path, consisting of a cycle with both positive and negative defined base literals; it is a *mixed cycle*. Because of this, the value of this path and the value of this justification for P are $undefined$. Therefore, the supported value of P in this justification is undefined, i.e. $SV_{\mathcal{I}}(P) = \mathbf{u}$. Normally, structures are two-valued (although they can be extended to three-valued structures). There is no two-valued well-founded interpretation $\mathcal{I}$ for Δ_2 such that $SV_{\mathcal{I}}(P) = P^{\mathcal{I}}$. Hence, this definition has no model. It is unsatisfiable.

   Note that because of the negative dependency of Q on P, you might be tempted to stratify it as follows.

$$\left\{\begin{array}{l} P \leftarrow Q \\ ------ \\ Q \leftarrow \neg P \end{array}\right\}$$

   But this is not a valid stratification because in layer1 P is defined in terms of Q, while Q itself is only defined later (layer2)! For a valid stratification it should hold that: all rules defining the same atom are in the same layer, and if an atom is defined in layer i,

   - the atoms on which it depends positively are defined in layer $1, \ldots, i$ (same layer or earlier),
   - the atoms on which it depends negatively are defined in layer $1, \ldots, i-1$ (strictly earlier).

   The given definition cannot be stratified in this way and hence is not valid and does not have a model.

3. For the third definition Δ_2, a similar explanation can be given as for definition Δ_3. Since its dependency graph contains a cycle with a negative edge, it cannot be stratified properly. It is has no two-valued well-founded interpretation where the value of each atom is equal to its supported value. Thus, the definition has no model.

3 FO(ID): semantics, default reasoning

Assignment Given the following definition, determine whether Coco and/or Skippy fly. Compute the model of the definition to find out (as always, use a graph representation to find out dependencies and stratify the definition).

$$\left\{\begin{array}{r l}
Fly(x) \leftarrow & Bird(x) \land \neg AbFly(x) \\
AbFly(x) \leftarrow & Ostrich(x) \land \neg AbOstrich(x) \\
AbOstrich(x) \leftarrow & SuperOstrich(x) \\
AbFly(x) \leftarrow & YoungSuperOstrich(x) \\
Bird(x) \leftarrow & Ostrich(x) \\
Bird(x) \leftarrow & Parrot(x) \\
Ostrich(x) \leftarrow & SuperOstrich(x) \\
SuperOstrich(x) \leftarrow & YoungSuperOstrich(x) \\
Parrot(Coco) \leftarrow & \\
YoungSuperOstrich(Skippy) \leftarrow &
\end{array}\right\}$$

Solution The answer is that Coco can fly and Skippy cannot.

We can stratify the definition as follows.

$$\left\{\begin{array}{r l}
Parrot(Coco) \leftarrow & \\
YoungSuperOstrich(Skippy) \leftarrow & \\
SuperOstrich(x) \leftarrow & YoungSuperOstrich(x) \\
AbOstrich(x) \leftarrow & SuperOstrich(x) \\
Ostrich(x) \leftarrow & SuperOstrich(x) \\
Bird(x) \leftarrow & Ostrich(x) \\
Bird(x) \leftarrow & Parrot(x) \\
-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,- & \\
AbFly(x) \leftarrow & Ostrich(x) \land \neg AbOstrich(x) \\
AbFly(x) \leftarrow & YoungSuperOstrich(x) \\
-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,-\,- & \\
Fly(x) \leftarrow & Bird(x) \land \neg AbFly(x)
\end{array}\right\}$$

This shows that this is a definition with a two-valued interpretation. We can then find the model of the definition using the constructive process using iterated rule application, as illustration graphically in 6. From this, we can construct the model by starting from the top and working downwards, each time only

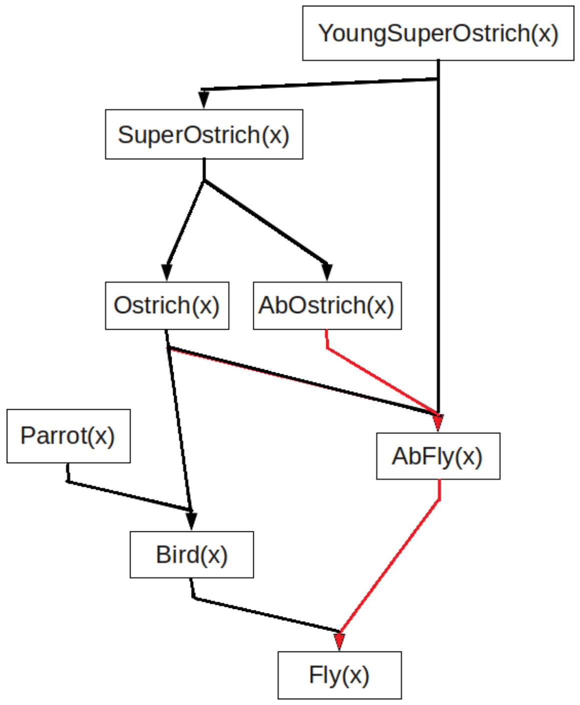

Figure 6: Graphical representation for construction of the well-founded model.

deriving information about a node when all parent nodes have been derived. So in the following order, information can be derived.

$$Parrot(Coco) \tag{1}$$

$$YoungSuperOstrich(Skippy) \tag{2}$$

$$SuperOstrich(Skippy) \tag{3}$$

$$AbOstrich(Skippy) \tag{4}$$

$$Ostrich(Skippy) \tag{5}$$

$$Bird(Skippy) \tag{6}$$

$$Bird(Coco) \tag{7}$$

$$AbFly(Skippy) \tag{8}$$

$$Fly(Coco) \tag{9}$$

We derived what is true in the model, what is not mentioned is false.

4 Packing

Assignment Given a rectangular area of a known dimension (width x height) and a set of squares, each of which has a known dimension, the problem is to pack all the squares into the rectangular area such that

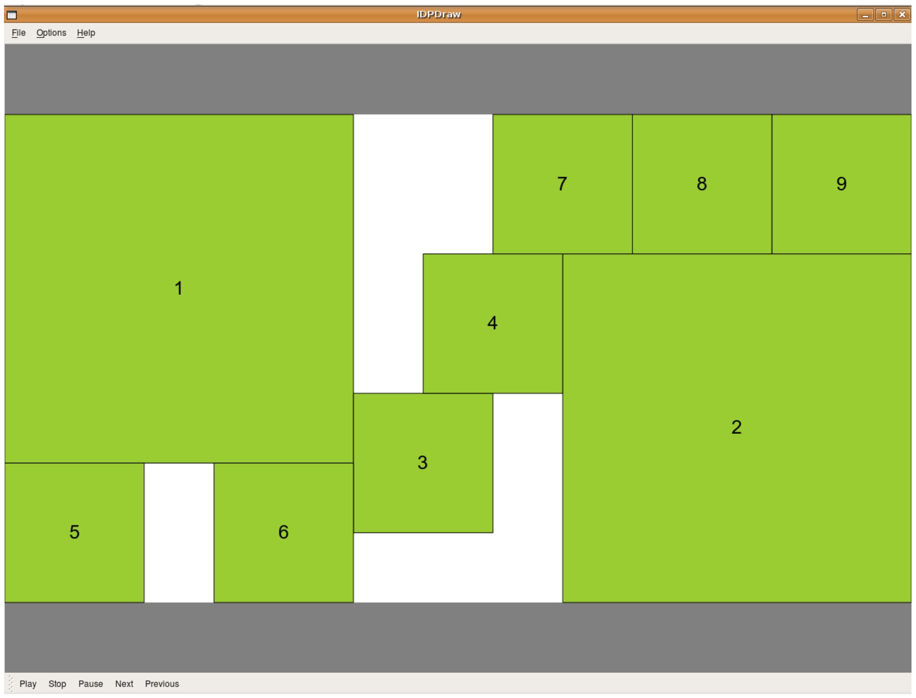

Figure 7: Graphical representation of one solution to the packing problem.

no two squares overlap. There can be empty spaces in the rectangular area. (For an example solution, see figure 7.)

The rectangular area forms a coordinate plane where the left upper corner of the area is the origin, the top edge is the x-axis, and the left edge is the y-axis. Suppose that we want to put the largest square in the top-left corner.

- Choose symbols for representing the position and size of squares and for representing the measurements of the total area. Choose a symbol for representing the largest block. Think about the necessary types.
- Give definitions for the following concepts:
  - a predicate $ValidPos : Square \rightarrow Bool$: indicates that the position of a square is within the bounds of the rectangular area.
  - predicates $LeftOf : (Square * Square) \rightarrow Bool$ and $Above : (Square * Square) \rightarrow Bool$ to specify relative positions of squares
  - a predicate $NoOverlap : (Square * Square) \rightarrow Bool$ which indicates that two square do not overlap
- Make a specification of correct packings as a FO(**.**) theory in this vocabulary.
- A set expression is of the form $\{(x_1, \ldots, x_n) : \phi\}$. Write a query (in this case, a set expression) to denote the set of all blocks above block A and not left of block B.
- Think of how to use the theory to solve at least two types of problems that require different forms of inference. (Note: you will probably not be able to solve this; this will be seen in lecture 5).

4.1 Solution

Vocabulary

- two types X and Y for specifying coordinates relative to a corner of the box (e.g. lowest left), using positive numbers
- two constants $Width, Height : () \to Int$ specifying the size of the total area
- a type $Square$, and a type $SizeType$ (containing positive numbers)
- a function $Size : Square \to SizeType$, to specify the size of a square (i.e., the length of a side)
- functions $XPos : Square \to X$ and $YPos : Square \to Y$ to specify the position of the corner of a square that is closest to (0,0)
- a constant $Largest : () \to Square$ to point to the largest square

Inductive definitions Defining what it means for a square to have a valid position:

$$\{\ ValidPos(sqr) \leftarrow (XPos(sqr) + Size(sqr) \leq Width()) \land (YPos(sqr) + Size(sqr) \leq Height()).\ \}$$

Defining the relative positions of squares:

$$\{LeftOf(sqr1, sqr2) \leftarrow XPos(sqr1) + Size(sqr1) \leq XPos(sqr2).\}$$

$$\{Above(sqr1, sqr2) \leftarrow YPos(sqr1) + Size(sqr1) \leq YPos(sqr2).\}$$

Defining the overlapping of squares in terms of their relative positions:

$$\left\{\begin{array}{c} NoOverlap(sqr, sqr) \leftarrow . \\ NoOverlap(sqr1, sqr2) \leftarrow LeftOf(sqr1, sqr2) \lor LeftOf(sqr2, sqr1) \land (Above(sqr1, sqr2) \lor Above(sqr2, sqr1)). \end{array}\right\}$$

A FO(.) theory Our specification for correct packings as a FO(**.**) theory in the chosen vocabulary contains the previously defined inductive definitions. We add sentences to the theory specifying that all squares have a valid position and that no two squares overlap.

$$\forall sqr : ValidPos(sqr).$$

$$\forall sqr1\ sqr2 : sqr1 \neq sqr2 \implies NoOverlap(sqr1, sqr2).$$

We also specify the largest square using a set expression. Note that there might be more than one square with the largest size. In the following expression, we assumed we then take the square with the smallest identifier. This requires the $Size$ and $Square$ types to have an order. We then specify that the largest square is in the top-left corner.

$$Size(Largest()) = max(lambdas : Size(s)).\ XPos(Largest()) = YPos(largest()) = 0.$$

A query using a set expression

$$\{sqr : Above(sqr, A) \wedge \neg LeftOf(sqr, B)\}$$

Two inferences Problems we might consider are:

- Given a packing decide if it is correct.
  Input:the theory, a structure *I* interpreting all symbols except the defined ones.
  Output: yes iff a model expanding *I* satisfies *T*.
  Required form of inference: model expansion (but a simple polynomial version, because all we need to do is to compute the definitions)
- Find a packing given box and squares.
  Input: the theory, and a structure *I* enumerating the types $Square$ and $SizeType$ and giving an interpretation to all symbols except XPos and YPos.
  Output: a new structure expanding *I* and satisfying *T*. It gives an interpretation to XPos and YPos, telling us how to pack the square in the rectangle.
  Required inference: model expansion
- Given a packing (assume it contains block 1): which blocks are above block 1?
  Required inference: Query $y : Above(y, 1)^M$ where M is the expansion of the database structure I of the previous point