Using Haskell for a Declarative Implementation of System Z Inference

Steven Kutsch
Faculty of Mathematics and Computer Science
University of Hagen
Germany
steven.kutsch@fernuni-hagen.de

Christoph Beierle
Faculty of Mathematics and Computer Science
University of Hagen
Germany
christoph.beierle@fernuni-hagen.de

Qualitative conditionals of the form ”if A then usually B” are a powerful means in knowledge representation, establishing a plausible relationship between A and B. When reasoning based on conditional knowledge consisting of a set of conditionals, a rich structure going beyond classical logic is required, e.g. ranking functions that assign a degree of implausibility to each possible world. System Z is a popular approach, using a unique partitioning of the knowledge base to generate the Pareto-minimal ranking function. This ranking function is used to answer questions plausibly based on the conditionals in the knowledge base. In this paper, we describe a Haskell implementation of system Z. To keep the Haskell code as close as possible to the formal definition of System Z, we make extensive use of language features such as list comprehension and higher order functions. For example, these are used to generate the required partition of the knowledge base or to represent the induced ranking function. The described system is used as a backend in the conditional reasoning tool InfOCF.

1 Introduction

Default rules of the form “If *A* then *usually / normally / preferably B*” play an important role in the area of knowledge representation and reasoning. They establish plausible relationships between *A* and *B*. A set of such rules can be used to represent the knowledge of a reasoning agent.

Example 1. *The following four sentences describe plausible relations in the domain of birds*

- *“Birds usually fly”*
- *“Penguins are usually birds”*
- *“Penguins usually don’t fly”*
- *“Birds usually have wings”*

A rational agent whose knowledge base is given by such a set of sentences should be able to reason and to draw inferences based on these sentences. While such knowledge bases may contain all relevant rules for an agent, they usually do not contain enough information to represent all plausible beliefs that a reasoning agent, operating based on this knowledge, should have. For instance, while believing that birds usually fly seems to be a direct consequence form the sentences given in Example 1, the situation is not so clear regarding e.g. the question whether penguins having wings usually do not fly. Thus, for a reasoning agent it is essential to extend a knowledge base to what is called a complete *epistemic state*, containing all beliefs necessary to answer arbitrary questions [7]. There are many ways to inductively complete a knowledge base and to represent the resulting epistemic state of an agent, e.g. using probability distributions [10], possibility theory [5], or ordinal conditional functions [13, 14]. These approaches assign a probability, possibility, or implausibility value to each possible world in order to be able to

Submitted to:
WFLP 2016

compare different possible worlds accordingly. Based on the induced ordering of the possible worlds, sentences as given in Example 1 can then be evaluated and used for inference.

An inference relation based on rules as given in Example 1 is nonmonotonic since the rules are not strict, but also allow for exceptions. System P is an axiomatic system providing a set of desirable properties for nonmonotonic inference relations [1]. It induces an inference relation by taking all models of a knowledge base into account. System Z [11] is an approach to relax this condition by defining an inference relation based on a single preferred model of a knowledge base. While there are other implementations of System Z such as `Z-log` [9] that focus on exploring the computational complexity of System Z, in this paper we present a declarative Haskell implementation of System Z called `HaskZ`. The main objective of our implementation that is to be close to the formal definition of the underlying concepts and algorithms. `HaskZ` also realizes system P inference, and supports experiments comparing the different inference relations.

The rest of this paper is organized in the following way. In Section 2 we recall the background of conditional logic, ranking functions and system Z as required here. in Section 3, we present a detailed overview of the implementation of `HaskZ`, and in Section 4 it is shown how `HaskZ` can be used. In Section 5, we conclude and point our further work.

2 Conditionals, Ranking Functions and System Z

Let $\mathcal{L}$ be a propositional language, generated by a finite set Σ of atoms $a,b,c,\ldots$. We denote formulas of $\mathcal{L}$ with uppercase letters $A,B,C,\ldots$. In formulas, we omit the *and*-connective, writing AB instead of $A \wedge B$. We indicate negation of a formula with overlining, i.e. $\overline{A}$ means $\neg A$. The set of possible worlds Ω contains all propositional interpretations over $\mathcal{L}$ these interpretations can easily be identified with the complete conjunctions over Σ. For $\omega \in \Omega$, $\omega \models A$ means that the propositional formula $A \in \mathcal{L}$ holds in the possible world ω.

2.1 Conditionals and Ranking Functions

To formalize the idea of plausible, probable, or possible connections between propositions, we introduce a new binary operator $|$ to form conditionals.

Definition 2 (Conditionals)**.** *Let $A,B \in \mathcal{L}$. Then $(B|A)$ is the* conditional *formalizing the conditional rule "if A then (usually) B". A is called the* antecedent *and B is called the* consequence*. The language of all conditionals over a propositional language $\mathcal{L}$ is denoted by $(\mathcal{L} \mid \mathcal{L})$.*

We will commonly use sets of conditionals as knowledge bases for our calculations.

Definition 3 (Conditional Knowledge Base)**.** *Let Σ be a propositional signature. A set*

$$\mathcal{R} = \{(B_1|A_1),\ldots,(B_n|A_n)\}$$

where every $A_i,B_i \in \mathcal{L}$ for $i \in \{1,\ldots,n\}$, is called a knowledge base*.*

Example 4 ($\mathcal{R}_{birds}$)**.** *We formalize the four sentences from Example 1 as conditionals.*

Let $\Sigma = \{b(birds), p(penguins), f(flying), w(wings)\}$. The knowledge base $\mathcal{R}_{birds} = \{r_1,r_2,r_3,r_4\}$ consists of the four conditionals:

$r_1 = (f|b)$ *"birds usually fly"*

$r_2 = (b|p)$ *"penguins are usually birds"*

$r_3 = \; (\overline{f}|p)$ *"penguins usually don't fly"*

$r_4 = \; (w|b)$ *"birds usually have wings"*

Conditionals are three-valued objects, which allows us to represent them as a *generalized indicator function* going back to [4]

$$(B|A)(\omega) = \begin{cases} 1 & \text{if} \quad \omega \models AB \qquad \text{(verification)} \\ 0 & \text{if} \quad \omega \models A\overline{B} \qquad \text{(falsification)} \\ u & \text{if} \quad \omega \models \overline{A} \qquad \text{(not applicable)} \end{cases} \tag{1}$$

In order to give appropriate semantics to conditionals, they are usually considered within richer structures such as *epistemic states* [7]. Beside certain (logical) knowledge, epistemic states also allow the representation of preferences, beliefs, assumptions of an intelligent agent. Basically, an epistemic state allows one to compare formulas or worlds with respect to plausibility, possibility, necessity, probability, etc.

Spohn's *ordinal conditional functions, OCFs* [12], also called ranking functions are capable of representing a complete epistemic state.

Definition 5 (ordinal conditional functions, OCFs)**.** *A* ordinal conditional function *is a function* $\kappa : \Omega \to \mathbb{N}$ *with* $\kappa^{-1}(0) \neq \emptyset$*.*

Ranking functions assign a degree of implausibility to every possible world. The higher $\kappa(\omega)$, the less plausible ω is considered by κ. Note that for each κ, at least one world must be most plausible, i.e. having rank 0. An OCF κ can be extended to arbitrary formulas $A \in \mathcal{L}$ by

$$\kappa(A) \quad = \quad \begin{cases} \min\{\kappa(\omega) \mid \omega \models A\} & \text{if } A \text{ is satisfiable} \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

and to conditionals $(B|A) \in (\mathcal{L} \mid \mathcal{L})$ by:

$$\kappa((B|A)) \quad = \quad \begin{cases} \kappa(AB) - \kappa(A) & \text{if } \kappa(A) \neq \infty \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

Note that $\kappa((B|A)) \geqslant 0$ since any ω satisfying AB also satisfies A and therefore $\kappa(AB) \geqslant \kappa(A)$.

Since ranking functions represent an epistemic state of a reasoning agent, we can define the acceptance of a conditional by an agent in epistemic state κ.

Definition 6 (Acceptance of Conditionals)**.** *Let κ be a ranking function. The conditional $(B|A)$ is accepted by κ, denoted by $\kappa \models (B|A)$, iff*

$$\kappa(AB) < \kappa(A\overline{B}). \tag{4}$$

Thus, a conditional is accepted iff its verification is considered strictly more plausible then its falsification.

We say that κ accepts a knowledge base $\mathcal{R}$, denoted by $\kappa \models \mathcal{R}$, iff $\kappa \models (B|A)$ for ever $(B|A) \in \mathcal{R}$. A knowledge base is *consistent*, iff a ranking function exists that accepts it [11].

Example 7. *Consider the knowledge base $\mathcal{R}_{birds}$ from Example 4. Table 1 shows a ranking function κ that accepts every conditional in $\mathcal{R}_{birds}$.*

Every ranking function induces a non-monotonic inference relation between formulas. This relation is based on the acceptance of conditionals in Definition 6.

Definition 8 (Ranking Function Inference)**.** *Let $A, B \in \mathcal{L}$ and κ a ranking function. Then B is a non-monotonic inference of A by κ, denoted by $A \mid\!\sim_\kappa B$, iff the conditional $(B|A)$ is accepted by κ.*

ω	$\kappa(\omega)$	ω	$\kappa(\omega)$	ω	$\kappa(\omega)$	ω	$\kappa(\omega)$
$bpfw$	2	$b\overline{p}fw$	0	$\overline{b}pfw$	2	$\overline{b}\overline{p}fw$	0
$bpf\overline{w}$	2	$b\overline{p}f\overline{w}$	1	$\overline{b}pf\overline{w}$	2	$\overline{b}\overline{p}f\overline{w}$	0
$bp\overline{f}w$	1	$b\overline{p}\overline{f}w$	1	$\overline{b}p\overline{f}w$	2	$\overline{b}\overline{p}\overline{f}w$	0
$bp\overline{f}\overline{w}$	1	$b\overline{p}\overline{f}\overline{w}$	1	$\overline{b}p\overline{f}\overline{w}$	2	$\overline{b}\overline{p}\overline{f}\overline{w}$	0

Table 1: Ranking functions accepting the knowledge base $\mathcal{R}_{birds}$ from example 4

2.2 System P and p-entailment

A common benchmark for non-monotonic inference relations is the axiom system P [1]. While the details of System P are not needed here, an important result is that it induces system P inference, called *p-entailment*, that coincides with the inference relation that takes every ranking function accepting a given knowledge base into account.

Definition 9 (p-entailment)**.** *[6] Let $A,B \in \mathcal{L}$ and $\mathcal{R}$ a knowledge base. Then B is p-entailed from A in the context of $\mathcal{R}$, denoted by $A \mid\!\sim^{\mathcal{R}}_{p} B$, iff $A \mid\!\sim_{\kappa} B$ for every κ accepting $\mathcal{R}$.*

Since a knowledge base is only consistent if a ranking function accepting it exists, this form of inference can be implemented by testing the consistency of the knowledge base augmented by the negated query conditional.

Proposition 10. *[6] Let $\mathcal{R}$ be a consistent knowledge base. Then*

$$A \mid\!\sim^{\mathcal{R}}_{p} B \quad \textit{iff} \quad \mathcal{R} \cup \big\{(\overline{B}|A)\big\} \textit{ is inconsistent.} \tag{5}$$

For checking the consistency of $\mathcal{R}$, a special partition of $\mathcal{R}$ based on the notion of *tolerance* can be used. Intuitively, a conditional r is tolerated by a set of conditionals $\mathcal{R}$, iff there is a world ω that satisfies r and does not falsify any $r' \in \mathcal{R}$ (as defined by (1)).

Definition 11 (Tolerance)**.** *[6] A conditional $(D|C)$ is* tolerated *by a knowledge base $\mathcal{R}$, iff there is a $\omega \in \Omega$ satisfying the formula*

$$CD \wedge \bigwedge_{(B|A)} (\overline{A} \vee B).$$

for every $(B|A) \in \mathcal{R}$.

Definition 12 (Ordered Partition)**.** *[6] Let $\mathcal{R}$ be a set of conditionals. $\mathcal{R}_p = (\mathcal{R}_0, \ldots, \mathcal{R}_k)$ is a ordered partition, iff $\{\mathcal{R}_0, \ldots, \mathcal{R}_k\}$ is a partition of $\mathcal{R}$ and for every $0 \leqslant i \leqslant k$, every $r \in \mathcal{R}_i$ is tolerated by the union $\bigcup_{j=i}^{k} \mathcal{R}_j$.*

The notion of order partition yields a consistency test.

Proposition 13. *[11] $\mathcal{R}$ is consistent, iff there is an ordered partition for $\mathcal{R}$.*

2.3 System Z

The condition for p-entailment is rather strict as it takes all ranking models of a knowledge base into account, possibly disallowing inferences that may still be considered plausible, although they do not hold in all ranking models of $\mathcal{R}$, but e.g. in a subset of preferred ranking models of $\mathcal{R}$. The idea

Listing 1: Algorithm to test for consistency of $\mathcal{R}$ (cf. [6]).

```
PROCEDURE: OrderedPartition
 INPUT  : Knowledge base R={(B_1|A_1),...,(B_n|A_n)}
 OUTPUT : Ordered partition (R_0,R_1,...,R_k) if R is consistent, NULL otherwise

 INT i:=0;
 WHILE(R ≠ ∅) DO
   R_i := { (B|A) ∈ R | R tolerates (B|A) };
   IF(R_i ≠ ∅){
   THEN
     R:=R \ R_i;
     i:=i+1;
   ELSE
     RETURN NULL; //R is inconsistent
 RETURN R_p = (R_0,...,R_k);
```

of system Z [11] is to define a plausible inference relation taking only a uniquely defined "best" or preferred model into account. For any consistent knowledge base $\mathcal{R}$, System Z defines a unique ranking function accepting $\mathcal{R}$. While in general, there are several different ordered partitions of $\mathcal{R}$, the procedure `OrderedPartition` in Algorithm 1 calculates the inclusion maximal ordered partition of $\mathcal{R}$, that is, every conditional is in the lowest possible subset.

Using the partition $\mathcal{R}_p = (\mathcal{R}_0,\ldots,\mathcal{R}_k)$ returned by `OrderedPartition`, the function $Z : \mathcal{R} \to \{0,\ldots,k\}$ is defined by

$$Z(r) = i \quad \text{iff} \quad r \in \mathcal{R}_i \tag{6}$$

With this function the System Z ranking function $\kappa^Z_{\mathcal{R}}$ is defined as [11]

$$\kappa^Z_{\mathcal{R}}(\omega) = \begin{cases} 0 & \text{iff } \omega \text{ does not falsify any } (B|A) \in \mathcal{R} \\ \max\limits_{(B|A)\in\mathcal{R}} \{Z((B|A))|\omega \models A\overline{B}\} + 1 & \text{otherwise.} \end{cases} \tag{7}$$

Because the ranking function $\kappa^Z_{\mathcal{R}}$ defined by System Z is based on the inclusion maximal partition satisfying the tolerance relation, it can be shown that it is the, with respect to assigned ranks, minimal ranking function accepting $\mathcal{R}$ [11].

Example 14. *The ranking function κ in Table 1 is the ranking function $\kappa^Z_{\mathcal{R}}$ using the inclusion-maximal ordered partition $\mathcal{R}_{birds} = (\{(f|b),(w|b)\},\{(b|p),(\overline{f}|p)\})$.*

While p-entailment takes all ranking functions of $\mathcal{R}$ into account, z-entailment is the inference relation induced by the ranking function $\kappa^Z_{\mathcal{R}}$.

Definition 15 (System Z inference; z-entailment)**.** *Let $A,B \in \mathcal{L}$ and $\mathcal{R}$ a knowledge base. Then B is z-entailed from A in the context of $\mathcal{R}$, denoted by $A \mid\!\sim^{\mathcal{R}}_{z} B$, iff $A \mid\!\sim_{\kappa^Z_{\mathcal{R}}} B$.*

The following example illustrates that z-entailment enables plausible inferences not possible with p-entailment.

Example 16. *A question we might want to answer based on the knowledge in $\mathcal{R}_{birds}$ is whether winged penguins are still unable to fly, that is whether from wp we can plausibly infer $\overline{f}$ in the context of $\mathcal{R}$, denoted by $wp \mid\!\sim^{\mathcal{R}} \overline{f}$.*

If we add the conditional $(f|wp)$*, representing the negation of the query conditional* $(\overline{f}|wp)$ *to* $\mathcal{R}_{birds}$*, the ordered partition* $(\{(f|b),(w|b)\},\{(b|p),(\overline{f}|p)\},\{(f|wp)\})$ *respects the tolerance condition. Therefore* $\mathcal{R}_{birds} \cup \{(f|wp)\}$ *is consistent and* $wp \not\mid\!\sim_p^{\mathcal{R}} \overline{f}$*.*

In contrast, using the ranking function $\kappa_{\mathcal{R}}^{Z}$ *listed in Table 1 we see that* $\kappa(p\overline{f}w) = 1 < 2 = \kappa(pfw)$ *and therefore* $wp \mid\!\sim_z^{\mathcal{R}} \overline{f}$*.*

3 Implementation

The implementation of `HaskZ` can be split into three parts. In the first part we will pay attention to the underlying datatypes that represent various formal parts of a logical language. The second part describes the implementation of the consistency check algorithm (Algorithm 1) and how it is used to implement p-entailment (Definition 9). The last section describes how $\kappa_{\mathcal{R}}^{Z}$ is calculated, represented, and used to realize z-entailment.

3.1 Logical Formulas and Knowledge Bases

The basic typeclass is `Atom`. Together with the type `Interpretation`, representing possible worlds, it is the basis of a logical system.

```
type Interpretation a = a -> Bool

class (Ord a) => Atom a where
  evalA :: a -> Interpretation a -> Bool
  printA :: a -> String
```

From this foundation, different types of formulas can be implemented. The typeclass `Formula` encapsulates these types of formulas and gives them a common interface.

```
class (Atom a) => Formula a f | f -> a where
  evalF :: f -> Interpretation a -> Bool
  printF :: f -> String
  getAtoms :: f -> [a]
```

In our implementation we need literals, conjunctions of literals, and formulas in disjunctive normal form (DNF), i.e. disjunctions of conjunctions. Any standardized representation of an arbitrary formula would work here. The decision for DNFs is mainly founded by compatability to other reasoning systems. For ease of modeling we represent conjunctions as lists of literals and DNFs as lists of conjunctions. All these types of formulas have suitable `Formula` instances.

```
data Literal a = Pos a
               | Neg a
               deriving (Eq, Ord)

type Conjunction a = [Literal a]

type DNF a = [Conjunction a]
```

Conditionals can not yet be expressed in this framework. We define them outside of this framework as pairs of formulas in disjunctive normal form and provide a three-valued datatype for evaluating them.

```
type Conditional a = (DNF a, DNF a)

data ConditionalIndicatorValue = Verified | Falsified | NotApplicable
```

```
                           deriving (Show, Eq)

evalConditional :: Atom a => Conditional a
                             -> Interpretation a
                             -> ConditionalIndicatorValue
evalConditional c w = case (evalF (fst c) w, evalF (snd c) w) of    -- (B|A)(w)
                        (True,True) -> Verified                     -- AB
                        (False,True) -> Falsified                   -- A!B
                        (_,False) -> NotApplicable                      -- !A
```

Note that the function `evalConditional` directly implements the *generalized indicator function* as given in (1).

This foundation of types is general enough to build many kinds of classical logical systems. In this paper, we only implement a propositional language by defining propositions as a type with a suitable `Atom` instance.

```
data Proposition = A | B | C | D | E | F | G | H | I | J | K | L | M
                 | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
                 | Prop String
                 deriving (Show, Eq, Ord)

instance Atom Proposition where
  evalA p i = i p
  printA (Prop s) = s
  printA p = map toLower $ show p
```

In this `Atom` instance, an interpretation `i` is just a function of type `(Proposition -> Bool)`.

We represent a knowledge base as a record type, containing all the necessary information.

```
class (Atom a) => KnowledgeBase k a | k -> a where
  name :: k -> String
  signature :: k -> [a]
  conditionals :: k -> [Conditional a]
  printKB :: k -> String

data PropositionalKnowledgeBase =
    PKB { pKBname :: String
        , pKBsignature :: [Proposition]
        , pKBconditionals :: [Conditional Proposition] }
```

Example 17. *The knowledge base $\mathcal{R}_{birds}$ from Example 4 is represented as:*

```
c1 = ([[Pos F]],[[Pos B]])
c2 = ([[Pos B]],[[Pos P]])
c3 = ([[Neg F]],[[Pos P]])
c4 = ([[Pos W]],[[Pos B]])

kb_birds = PKB { pKBname = "birds"
               , pKBsignature = [B,P,F,W]
               , pKBconditionals = [c1, c2, c3,c4] }
```

Based on an actual knowledge base with a fixed and finite signature, we can generate the set Ω of all possible worlds as a finite list of functions, making use of the bijection between complete conjunctions in Ω and functions of the type $\omega : \Sigma \to Bool$. These functions make use of the *closed world assumption* and assign `False` to every atom not in the signature. The function `bigOmega` generates this list of functions, by generating all possible combinations of `True` and `False` of length $|\Sigma|$ and constructing a closure for every combination using the function `omega`. This closure realizes a lookup, returning the boolean value of the argument in this interpretation and `False` if the argument is not part of the signature.

```
omega :: (Eq a) => ([Proposition],[Bool]) -> Interpretation a
omega w = (\a -> case elemIndex a (fst w) of
                   Just x -> (snd w) !! x
                   Nothing -> False) -- closed world assumption

bigOmega :: PropositionalKnowledgeBase -> [Interpretation Proposition]
bigOmega kb = map (\x -> omega (props,x)) $
                  combinations propcount [False, True]
  where props = signature kb
        propcount = length props

combinations :: (Num a, Ord a) => a -> [b] -> [[b]]
combinations n _ | n <= 0 = [[]]
combinations 1 xs = map (:[]) xs
combinations n xs = (:) <$> xs <*> combinations (n-1) xs
```

3.2 Consistency Check and System P

To implement the consistency check detailed in Algorithm 1 we need the tolerance relation between a conditional and a knowledge base. We implement this relationship using two nested list comprehensions which immediately follow from Definition 11.

```
tolerated :: Conditional Proposition -> PropositionalKnowledgeBase -> Bool
tolerated c kb =
    not $ null [w | w <- bigOmega kb, evalConditional c w == Verified,
                    null [c' | c' <- conditionals kb,
                               evalConditional c' w == Falsified] ]
```

We use the function `bigOmega` to generate the list of possible worlds as described above. If $c = (B|A)$, then the generated list is empty exactly when there is no $\omega \in \Omega$ for which $\omega \models AB$ and $\omega \not\models C\overline{D}$ for any $c' = (D|C) \in \mathcal{R}$.

Using the function `tolerated` we can implement `OrderedPartition` (Algorithm 1) as a recursive function `orderedPartition`. Since the Algorithm returns `NULL` if the knowledge base is inconsistent, we use the `Maybe` type to handle failure.

```
orderedPartition :: PropositionalKnowledgeBase
                         -> Maybe [[Conditional Proposition]]
orderedPartition kb = fmap reverse (pkb [] $ conditionals kb)
  where pkb parts [] = Just parts
        pkb parts l = let tcs = toleratedcs l
                      in if (tcs == [])
                         then Nothing      -- KB inconsistent
                         else pkb (tcs:parts) $ l \\ tcs
        toleratedcs l =
             [c | c <- l
                , tolerated c (defaultPKB { pKBsignature = signature kb
                                           , pKBconditionals = l })]
```

The locally defined function `toleratedcs` uses a list comprehension to construct the sublist only containing conditionals tolerated by the original list (line 7 in Algorithm 1). It is necessary to use the function `defaultPKB` to construct a knowledge base matching the type of `tolerated`. The function `pkb` handles the bookkeeping such as the construction of the list of sublists and the actual recursion.

It then returns the constructed ordered partition, or `Nothing` if there are no conditionals left that are tolerated by the knowledge base. The resulting list of sublists needs to be reversed to fit the output of `OrderedPartition` in Algorithm 1.

Since this algorithm realizes a consistency test for knowledge bases, we use it to implement p-entailment from Definition 9, by testing the consistency of the knowledge base after adding the negated query conditional.

```
p_entails :: PropositionalKnowledgeBase -> DNF Proposition
             -> DNF Proposition -> Bool
p_entails kb ant con = isNothing $ orderedPartition kb'
  where kb' =
    defaultPKB { pKBsignature = (signature kb),
                 pKBconditionals = ((negateDNF con),ant) : (conditionals kb) }
```

3.3 System Z

The computation of the unique minimal model of the knowledge base $\mathcal{R}$ is at the core of System Z. The higher order function `calcZ` uses the function `orderedPartition` to generate the function $Z : \mathcal{R} \to \mathbb{N}$ as defined in (6).

```
calcZ :: PropositionalKnowledgeBase -> (Conditional Proposition -> Int)
calcZ kb = case orderedPartition kb of
             Just parts -> (\c -> fromJust $ findIndex (elem c) parts)
             Nothing -> error "KB inconsistent"
```

The returned function uses `findIndex` to determine the index of the partition containing its argument. Using the function `fromJust` is save in this case, since the only function using the returned function is guaranteed to only pass it conditionals contained in the original knowledge base and therefore also contained in one of the sublists in the result of `orderedPartition`. The use of `error` in the case of an inconsistent knowledge base is justified by the use case of `HaskZ` detailed in Section 4. `HaskZ` can be used in an interactive session, where the error is simply printed as a message, or as a backend to another program, that expects an error code in the case of an inconsistent knowledge base.

The function `kappa_z` generates the ranking function $\kappa^Z_{\mathcal{R}} : \Omega \to \mathbb{N}$ according to (7).

```
kappa_z :: PropositionalKnowledgeBase -> (Interpretation Proposition -> Int)
kappa_z kb = (\w -> if all notFalsified $ condStruct w
                    then 0
                    else maximum [z c | c <- falsifiedCs w] + 1)
  where
    z = calcZ kb
    condStruct w = map ((flip evalConditional) w) $ conditionals kb
    falsifiedCs w = [c | c <- conditionals kb
                       , evalConditional c w == Falsified]
```

The returned function models the formal definition of a ranking function closely, since ranking functions are defined as functions between interpretations and positive integers, cf. Definition 5. Using this function we can implement z-entailment $A \mid\!\sim^{\mathcal{R}}_{Z} B$, i.e. checking whether A entails B in the context of the knowledge base $\mathcal{R}$ using the unique ranking function $\kappa^Z_{\mathcal{R}}$. We realize this relationship by implementing Definition 6.

```
z_entails :: PropositionalKnowledgeBase
             -> DNF Proposition -> DNF Proposition -> Bool
z_entails kb ant cons = min_kappa verifyingWorlds < min_kappa falsifyingWorlds
```

```
  where worlds = bigOmega kb
        kappa = kappa_z kb
        verifyingWorlds = [w | w <- worlds
                             , evalF ant w
                             , evalF cons w]
        falsifyingWorlds = [w | w <- worlds
                              , evalF ant w
                              , evalF (negateDNF cons) w]
        min_kappa l = minimum $ map kappa l
```

We use list comprehensions to determine the worlds that verify or falsify the conjunction of the antecedence and the consequence. From those we select the minimal κ-value. If the minimal rank of the verifying worlds is smaller then the minimal rank of the falsifying worlds, the inference `ant` $\mid\!\sim_{\kappa^Z_{\mathcal{R}}}$ `cons` holds.

In the implementation of `HaskZ` we make heavy use of features like list comprehensions and higher order functions to stay close to the formal definitions. List comprehensions are used to construct lists of objects that have the properties required by the definitions. We model interpretations as functions from signatures to boolean values and ranking functions as functions from interpretations to positive integers. All of this helps to see the close connections between runnable code and formal definition, and it makes arguing about its correctness easy.

4 Using `HaskZ`

There are two ways of using `HaskZ`. It can be used for interactive experiments in a `ghci` session by importing the relevant modules or as a backend that writes results to files in machine readable form. This section details the work flow in both cases.

4.1 `HaskZ` in `ghci`

We start with a file named `birds.hs` containing the knowledge base $\mathcal{R}_{birds}$ from Example 17 together with the imports of modules containing the needed functionality.

```
import Data.Logic.SystemP
import Data.Logic.SystemZ
import Text.PrettyPrint.TruthTable

c1 = ([[Pos F]],[[Pos B]])
c2 = ([[Pos B]],[[Pos P]])
c3 = ([[Neg F]],[[Pos P]])
c4 = ([[Pos W]],[[Pos B]])

kb = PKB { pKBname = "birds"
         , pKBsignature = [B,P,F,W]
         , pKBconditionals = [c1, c2, c3,c4] }
```

For convenient use of the functions `p_entails` and `z_entails` we can define operators already containing the knowledge base `kb`.

```
(|~p) = p_entails kb
(|~z) = z_entails kb
```

Loading this file, after installing HaskZ with cabal[1], in a ghci-session, allows us to perform several experiments. We can calculate the ranking function $\kappa^Z_{\mathcal{R}}$ using the function printTruthTable.

```
> printTruthTable kb
b p f w|( f | b )|( b | p )|( !f | p )|( w | b )|kappa_z
-------|---------|---------|----------|---------|-------
0 0 0 0|    u    |    u    |     u    |    u    |   0
0 0 0 1|    u    |    u    |     u    |    u    |   0
0 0 1 0|    u    |    u    |     u    |    u    |   0
0 0 1 1|    u    |    u    |     u    |    u    |   0
0 1 0 0|    u    |    -    |     +    |    u    |   2
0 1 0 1|    u    |    -    |     +    |    u    |   2
0 1 1 0|    u    |    -    |     -    |    u    |   2
0 1 1 1|    u    |    -    |     -    |    u    |   2
1 0 0 0|    -    |    u    |     u    |    -    |   1
1 0 0 1|    -    |    u    |     u    |    +    |   1
1 0 1 0|    +    |    u    |     u    |    -    |   1
1 0 1 1|    +    |    u    |     u    |    +    |   0
1 1 0 0|    -    |    +    |     +    |    -    |   1
1 1 0 1|    -    |    +    |     +    |    +    |   1
1 1 1 0|    +    |    +    |     -    |    -    |   2
1 1 1 1|    +    |    +    |     -    |    +    |   2
-------|---------|---------|----------|---------|-------
   Z(r)|    0    |    1    |     1    |    0    |
```

The result is produced using the boxes-library[2]. It lists every possible world in the first column, followed by a column showing the Conditional Indicator Value (+ = verified, - = falsified, u = not applicable) of every conditional in the knowledge base. The bottom line lists the value of the function Z for every conditional, and the last column shows the calculated ranking function $\kappa^Z_{\mathcal{R}}$.

Using the two operators for inference we can answer the queries from Example 16 in the context of the knowledge base using the different semantics.

```
> [[Pos P, Pos W]] |~p [[Neg F]]
False
> [[Pos P, Pos W]] |~z [[Neg F]]
True
```

4.2 HaskZ as a backend

Currently, HaskZ is used as a backend in a conditional reasoning tool called InfOCF, that produces files like birds.hs. These files also contain a main function that either writes some result to a file that can be read by InfOCF or, in the case of inference, terminates with a return code to indicate the inference result.

If we add the line

```
main = exportOCF kb
```

to the end of the file birds.hs and run the program using runhaskell, the following output is written to the file birds_systemz.ocf.

```
p,b,f,w
0,0,0,0;0
```

[1] www.haskell.org/cabal

[2] hackage.haskell.org/package/boxes

```
0,0,0,1;0
0,0,1,0;0
0,0,1,1;0
0,1,0,0;1
0,1,0,1;1
0,1,1,0;1
0,1,1,1;0
1,0,0,0;2
1,0,0,1;2
1,0,1,0;2
1,0,1,1;2
1,1,0,0;1
1,1,0,1;1
1,1,1,0;2
1,1,1,1;2
```

This file is then read by `InfOCF` and interpreted as a ranking function that can be compared with other ranking functions produced by different backends.

To get the return code indicating the result of a query, we use either `p_entails_rc` or `z_entails_rc` as our definition of main.

```
p_entails_rc :: PropositionalKnowledgeBase -> DNF Proposition
                -> DNF Proposition -> IO ()
p_entails_rc kb ant con = if p_entails kb ant con
                          then exitSuccess
                          else exitFailure

Z_entails_rc :: PropositionalKnowledgeBase -> DNF Proposition
                -> DNF Proposition -> IO ()
Z_entails_rc kb ant con = if z_entails kb ant con
                          then exitSuccess
                          else exitFailure
```

The return code is read by `InfOCF`, when it can be further processed depending on the calling function

5 Conclusions

We presented the declarative Haskell implementation of System Z called `HaskZ`. It makes use of high level functional and declarative programming techniques to keep the executable code close to the formal definitions. These features make it easy to follow the code based on the formal definitions and help to convince the programmer and the user of the correctness of the implementation.

The foundational type definitions make it easy to formulate knowledge bases by hand and through automated code generation. This makes `HaskZ` usable as a experimentation environment and as a backend to systems like `InfOCF` that also provide other nonmonotonic inference relations, e.g. based on c-representations [8, 2, 3].

We plan to extend the foundational types to a more general framework for representing further logics and additional inference relations in Haskell.

References

[1] Ernest W. Adams (1975): *The Logic of Conditionals: An Application of Probability to Deductive Logic*. Synthese Library, Springer Science+Business Media, Dordrecht, NL.

[2] C. Beierle, C. Eichhorn & G. Kern-Isberner (2016): *Skeptical Inference Based on C-representations and its Characterization as a Constraint Satisfaction Problem*. In M. Gyssens & G. R. Simari, editors: *Foundations of Information and Knowledge Systems - 9th International Symposium, FoIKS 2016, Linz, Austria, March 7-11, 2016. Proceedings*, *LNCS* 9616, Springer, pp. 65–82.

[3] C. Beierle, C. Eichhorn, G. Kern-Isberner & S. Kutsch (2016): *Skeptical, Weakly Skeptical, and Credulous Inference Based on Preferred Ranking Functions*. In: *Proceedings 22nd European Conference on Artificial Intelligence, ECAI-2016.* (to appear).

[4] B. DeFinetti (1974): *Theory of Probability*. 1,2, John Wiley & Sons.

[5] Didier Dubois & Henry Prade (2015): *Possibility Theory and Its Applications: Where Do We Stand?* In Janusz Kacprzyk & Witold Pedrycz, editors: *Springer Handbook of Computational Intelligence*, Springer Berlin Heidelberg, Berlin, DE, pp. 31–60.

[6] M. Goldszmidt & J. Pearl (1996): *Qualitative probabilities for default reasoning, belief revision, and causal modeling*. *Artificial Intelligence* 84, pp. 57–112.

[7] J.Y. Halpern (2005): *Reasoning About Uncertainty*. MIT Press.

[8] G. Kern-Isberner (2001): *Conditionals in nonmonotonic reasoning and belief revision*. *LNAI* 2087, Springer, Lecture Notes in Artificial Intelligence LNAI 2087.

[9] Michael Minock & Hansi Kraus (2002): *Z-log: Applying System-Z*. In: *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02)*, pp. 545 – 548.

[10] Judea Pearl (1988): *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[11] Judea Pearl (1990): *System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning*. In Rohit Parikh, editor: *Proceedings of the 3rd conference on Theoretical aspects of reasoning about knowledge (TARK1990)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 121–135.

[12] W. Spohn (1988): *Ordinal conditional functions: a dynamic theory of epistemic states*. In W.L. Harper & B. Skyrms, editors: *Causation in Decision, Belief Change, and Statistics, II*, Kluwer Academic Publishers, pp. 105–134.

[13] Wolfgang Spohn (1988): *Ordinal Conditional Functions: A Dynamic Theory of Epistemic States*. In: *Causation in Decision, Belief Change and Statistics: Proceedings of the Irvine Conference on Probability and Causation*, *The Western Ontario Series in Philosophy of Science* 42, Springer Science+Business Media, Dordrecht, NL, pp. 105–134.

[14] Wolfgang Spohn (2012): *The Laws of Belief: Ranking Theory and Its Philosophical Applications*. Oxford University Press, Oxford, UK.