Synthesising and Implementing Tableau Calculi for Interrogative Epistemic Logics

Ştefan Minică
Amstelveen, The Netherlands
stefan.minica@gmail.com

Mohammad Khodadadi, Renate A. Schmidt, Dmitry Tishkovsky*
The University of Manchester, Manchester, UK
khodadadi,dmitry,schmidt@cs.man.ac.uk

Abstract

This paper presents a labelled tableau approach for deciding interrogative-epistemic logics (IEL). Tableau calculi for these logics have been derived using a recently introduced tableau synthesis method. We also consider an extension of the framework for a setting with questioning modalities over sequences of formulae called sequential questioning logic (SQL). We have implemented the calculi using two approaches. The first implementation has been obtained with the tableau prover generation software MetTeI^2, while the other implementation is a prover implemented in `Haskell`.

1 Introduction

The paper focusses on developing and implementing automated reasoning tools for interrogative epistemic logics (IEL). Interrogative or erotetic logics have a long tradition alongside declarative and epistemic logics. Interrogative Epistemic Logic (henceforth, IEL) also referred to as DELQ (for Dynamic Epistemic Logic of Questions), enriches a standard multi-agent epistemic modal logic with interrogative components [15, 7]. Intuitively this is done by adding an "issue" relation over a set of possible worlds. This relation is meant to represent structural changes brought about by dynamic actions of raising and answering questions. Besides the standard epistemic modality the logic also introduces a static modality over the issue relation. This gives rise to interaction between the epistemic and interrogative components. Such aspects are captured by an intersection modality which is then used to describe how dynamic questioning effects depend on the structure of the issues raised and previous knowledge.

A second addition are the dynamic actions that express interrogative or epistemic events explicitly in the language. Their effect is to change the interrogative and epistemic states and to add more structure to the existing issue or epistemic relations.

While automated reasoning tools are widespread for declarative and epistemic modal logics, for interrogative epistemic logics there are currently no implemented automated reasoning systems. The usefulness of automating reasoning for other logics, such as epistemic modal logics, has been proven already by many applications. Very often in epistemic scenarios obtaining the relevant information is an essential part. Adding an interrogative component makes modelling and reasoning about obtaining relevant information possible.

For dynamic epistemic logics (DEL), automated reasoning tools focused so far on solving model-checking tasks [17]. Other tableau-based provers for modal logics, e.g., [3], incorporate dynamic modalities for informative actions, e.g., public announcements [4, 2]. However, none of the existing software tools contain questioning modalities and moreover, none of them offer a generic method to generate a prover for a logic starting from a semantic specifications.

*This research is supported by UK EPSRC research grant EP/H043748/1.

Planning in contexts involving interaction between questions and knowledge is reducible to testing validities of interrogative epistemic logic. However, the existing proof systems for dynamic epistemic logics [4, 14] are not fully automated. They are usually Hilbert-style calculi in which formulae with non factual content have special substitution rules.

A longstanding problem for dynamic logics is the fact that they are not closed under uniform substitution, and therefore, they are not suitable for an algebraic treatment and do not lend themselves well to automatic reasoning techniques. Previous research in this area focused on identifying substitution closed fragments of such logics, which can still preserve some of the features that have made dynamic logics so successful for modelling information exchange.

The approach of this paper gives an alternative solution based on first translating the semantics of the modal language into many-sorted first-order logic and then turning it into a tableau calculus for the corresponding first-order fragment. Based on this tableau calculus two tableau based reasoning tools have been developed for interrogative-epistemic logics.

The paper is structured as follows. We start in Section 2 by introducing the details of IEL. Then we apply the tableau synthesis framework to IEL in Section 3. In Section 4 we present an extended logic, called SQL, which uses sequences of formulae inside the dynamic modalities. We continue in Section 5 with introducing and discussing the MetTeL2 implementation for IEL. In Section 6 we present and discuss the `Haskell` implementation `Qtab.lhs` for IEL which also illustrates the extension to questioning sequences. We draw some conclusions in the final section. Further implementation details and illustrative code output, which could not be included due to space limitations, can be found in the long version of the paper [8].

2 Interrogative Epistemic Logics

The approach of IEL [15, 7] starts by enriching a standard multi-agent epistemic modal logic with interrogative components. This is done in two stages. The first addition consists of a static modality over an "issue" relation. The intuitive meaning of this modality is close to the traditional epistemic notion, but instead of representing actual knowledge it stands for what the agents would like to find out. It represents future epistemic goals that are expressed by asking questions and will eventually be achieved by obtaining answers.

For technical reasons a third modality, expressing the interaction between the epistemic and the interrogative components is also introduced using the intersection of the standard epistemic relation and the newly introduce issue relation. This also has an intuitive meaning that goes beyond the traditional epistemic notion. It expresses how the future knowledge depends on both the current epistemic state of the agent and the epistemic goals guiding the ongoing questioning dynamics. This is what the agent will come to know if all the questions he raised so far would be answered one way or another.

In this paper nominals are added to the language alongside propositions. The second addition consists of two dynamic modalities, one for questioning actions or queries and one for answering actions or resolution actions. Intuitively, such modalities change the underlying structures by refining their component relations.

A formula φ in the language of IEL is defined by the following BNF:

$$\varphi \quad ::= \quad n \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid [Q]\varphi$$

Here, n, p, a denoting nominals, propositions, and agent labels, respectively. $\Box$ stands for modalities, that is $\Box \in \{Q_a, X_a, K_a\}$, respectively being static questioning, interaction, and epistemic modalities. Finally, Q stands for actions that change the underlying models, that is

$Q \in \{\varphi?_a, !_a\}$, representing dynamic questioning actions and resolution actions, respectively. This language is meant to express minimal interrogative-epistemic facts. The @ operator, which is a standard addition for hybrid logics is introduced later in the specification language where it is useful for both internalizing the semantics and formulating labeled tableau rules.

The language can express the interaction between questions and information in two ways. First, by using a (static) intersection modality $X_a\varphi$. Second, through the dynamic modalities $[Q]$, encoding model-changing operations by means of questioning and resolution actions.

The logic has a standard modal semantics over issue-epistemic models, $M = \langle W, \overset{a}{\approx}, \overset{a}{\sim}, V\rangle$. When used inside a tuple representing a model $\overset{a}{\approx}$ and $\overset{a}{\sim}$ are meant as shorthand notations for $(\overset{a}{\approx})_{a\in A}$ respectively $(\overset{a}{\sim})_{a\in A}$ for A the set of all agent labels. We use the expected Boolean clauses and the usual relational (modal) clauses involving $\overset{a}{\approx}$ for Q_a and $\overset{a}{\sim}$ for K_a. We also use equivalence relations for $\approx$ and $\sim$ throughout the paper. However, if needed, the framework can be generalized to other structures by correspondingly changing the background theory.

The intersection modality X_a is defined using $\overset{a}{\approx}\cap\overset{a}{\sim}$ as follows:

$$M \models_w X_a\varphi \quad \text{iff} \quad \forall v \in W : w\,(\overset{a}{\approx}\cap\overset{a}{\sim})\,v \Rightarrow M \models_v \varphi$$

Dynamic modalities express model changing operations of asking and resolution:

$[\varphi?]_a\,\psi$ "after φ is asked, ψ is the case" $M_? = \langle W, \overset{a}{\approx}_?, \overset{a}{\sim}, V\rangle$; $\overset{a}{\approx}_? = \overset{a}{\approx}\cap\overset{\varphi}{\equiv}_M$

$[!]_a\varphi$ "after having answered the questions raised, φ is true"
$M_! = \langle W, \overset{a}{\approx}, \overset{a}{\sim}_!, V\rangle$; $\overset{a}{\sim}_! = \overset{a}{\sim}\cap\overset{a}{\approx}$

Here, $\overset{\varphi}{\equiv}_M = \{(w,v) \mid ||\varphi||^M_w = ||\varphi||^M_v\}$ is the set of M-world pairs in which φ has the same truth value. The intuitive reading for a questioning action represented by the modality $[\varphi?]_a\psi$ is that of splitting the domain into φ worlds and non-φ worlds, by raising a question, or by making φ an issue. The intuitive reading for the resolution action represented by the $[!]_a\varphi$ modality is to add new knowledge by refining the epistemic relation in such a way that afterwards all the issues raised so far are solved. Indexing the actions with agent labels can also model privacy in questioning or can be used to add agent-specific preconditions for question execution. However, we assume our actions to be 'public' and 'preconditionless', i.e., they affect the epistemic/questioning states for all agents, and they do not require any conditions for execution. For this reason, indexing the modalities with agent labels will only play a genuine role in this paper for the static part of the logic.

The language of IEL has two parts. The static part is a hybrid modal logic with nominals and intersection. The dynamic part adds the dynamic modalities capturing model changing operations. The *static part* of the logic is axiomatised by a customary hybrid logic system [12] with nominals, S5 axioms for $\sim$ and $\approx$, and an intersection axiom for static resolution expressed by the following pure formula:

$$\widehat{K}_a i \wedge \widehat{Q}_a i \leftrightarrow \widehat{X}_a i, \qquad \text{where } i \text{ is a nominal.}$$

Here, $\widehat{Q}_a, \widehat{K}_a$ and $\widehat{X}_a$ are the diamond modalities defined as the duals of the box modalities Q_a, K_a, X_a introduced before.

The *dynamic part* of IEL introduces modalities which change the underlying static models. The logical behaviour of this new kind of connectives is captured by reduction axioms. These describe the relation between the underlying static structures before and after a questioning action or resolution action takes place. Formulas containing dynamic modalities can be reduced

to equivalent static formulas using reduction axioms like the following ones (for $b \in \{n, p\}$, $\Box \in \{Q, X\}$ and $q \in \{\varphi?, !\}$):

$$[q]_a\, b \leftrightarrow b, \qquad [q]_a \neg\psi \leftrightarrow \neg[q]_a\psi, \qquad [q]_a(\psi \wedge \chi) \leftrightarrow [q]_a\psi \wedge [q]_a\chi \tag{1a}$$

$$[!]_a \Box_a \psi \leftrightarrow \Box_a [!]_a \psi, \quad [!]_a K_a \varphi \leftrightarrow X_a [!]_a \varphi, \quad [\varphi?]_a K_a \psi \leftrightarrow K_a [\varphi?]_a \psi \tag{1b}$$

$$[\varphi?]_a Q_a \psi \leftrightarrow (\varphi \wedge Q_a(\varphi \rightarrow [\varphi?]_a\psi)) \vee (\neg\varphi \wedge Q_a(\neg\varphi \rightarrow [\varphi?]_a\psi)) \tag{1c}$$

$$[\varphi?]_a X_a \psi \leftrightarrow (\varphi \wedge X_a(\varphi \rightarrow [\varphi?]_a\psi)) \vee (\neg\varphi \wedge X_a(\neg\varphi \rightarrow [\varphi?]_a\psi)) \tag{1d}$$

This treatment is in line with the generic DEL methodology introduced in [4, 14] extended to include an additional interrogative component. Further technical details, possible extensions and examples of applications of IEL can be found in [7, 15].

We start from IEL as minimal logic when synthesizing and implementing the tableau calculus. Several extensions of the framework that handle various levels of privacy for epistemic-questioning actions can be added in a modular way using the same general synthesis method.

- Multi-agent questioning preconditions
- Group-opaque dynamic questioning effects
- Epistemic indistinguishability in questioning
- Dynamic questioning sequences

Due to lack of space in this paper we will discuss in detail only the last extension.

The extension requires questioning actions of a more complex type capable to model modalities over sequences of questions not just one formula. We briefly motivate now why such an extension is desirable and useful. The concrete details of the extension method are introduced later in Section 4, after all the needed details of the specification language are introduced.

Note that the standard IEL reduction axioms do not have cases for iterated modalities. Indeed, such cases are not, at some level of abstraction, necessary since they can be dealt with logically "from inside out". For any formulae φ, ψ, χ of IEL, $[\varphi?][\psi?]\chi$ can be dealt with in the following order, given the recursive structure of the reduction axioms:

$$[\varphi?][\psi?]\chi \leftrightarrow [\varphi?](\mathrm{trs}([\psi?]\chi)) \leftrightarrow \mathrm{trs}([\varphi?](\mathrm{trs}([\psi?]\chi)))$$

Here trs stands for the translation of the right side in the reduction axioms from Equation 1, as defined in Equation 4. While such a rule of thumb can be useful to some extent for human reasoning it is nevertheless not optimal for automatic reasoning. Even though we left out iterated modalities during the exposition of the logic we later deal with them as the implementation details require them. For this a direct recursion over the formulae would be optimal, and we approach this aspect in both of our implementations in Sections 5 and 6. Here we only briefly discuss some of the available modelling options.

One possible way to achieve this is by directly reducing iterated modalities to an equivalent non-iterated dynamic modality and then use the existing reduction axioms. For instance, for public announcement logic (PAL), which uses world-elimination instead of link-cutting, iteration of dynamic modalities boils down to the following equivalence for announcement composition:

$$[!\varphi][!\psi]\chi \leftrightarrow [!(\varphi \wedge [!\varphi]\psi)]\chi$$

However, there can be no such nor similar equivalent for IEL without sequences:

No single question can induce a 4-equivalence-classes partition.

All these complications are avoided by a language with questioning sequences:

$$[\varphi?][\psi?]\chi \leftrightarrow [\langle \varphi, \psi \rangle ?]\chi$$

This can be achieved in a modular way by keeping the syntax and the semantics of the language unchanged for both the static part and the dynamic resolution part and replacing our initial questioning modality with a modality defined over sequences of questions. This is also a dynamic modality for the collective action of asking questions or raising issues but the syntax uses a list of formulae $\sigma = \langle \varphi_0, \ldots, \varphi_n \rangle$:

$$[\sigma?]\,\varphi \quad \text{“after the questions in } \sigma \text{ are asked, } \varphi \text{ is the case”}.$$

The semantic definition of the dynamic modality is changed accordingly, the action's effect is to change an initial model M into a new model $M_{\sigma?} = \langle W, \overset{x}{\approx}_?, \overset{a}{\sim}, V \rangle$ with:

$$\overset{x}{\approx}_? = \overset{x}{\approx}_{\langle\rangle ?} \cap \bigcap_{n=0}^{|\sigma|-1} \overset{\varphi_n}{\equiv}_{M_n} \text{ for any agent } x.$$

For any model M and questioning sequence $\sigma = \langle \varphi_0, \ldots, \varphi_n \rangle$, the model M_k denotes the model obtained after a questioning action using the sequence $\sigma_k = \langle \varphi_0, \ldots, \varphi_k \rangle$ for $0 \leq k \leq n$. An empty questioning sequence does not change a model: $M_{\langle\rangle ?} = M$ and, in particular, $\approx_{\langle\rangle ?} = \approx$. This allows one to deal with longer sequences recursively using a head-tail pattern:

$$[\langle\rangle ?]\varphi \leftrightarrow \varphi \text{ and } [\langle \varphi_0, \varphi_1, \ldots, \varphi_n \rangle ?]\varphi \leftrightarrow [\langle \varphi_0 \rangle ?][\langle \varphi_1, \ldots, \varphi_n \rangle ?]\varphi.$$

The case of iterating the resolution modality $[!]$ is much simpler because sequences of any length $[\langle !, !, \ldots \rangle]$ can be collapsed to a resolution sequence of length one $[\langle ! \rangle]$. This is so because the resolution modality is idempotent: $!;! = !$. Therefore, we only have to consider iteration between sequential asking modalities and single, i.e., depth one, resolution modalities.

This leads to the more general reduction axioms we introduce in Section 4. The reduction axioms in Equation 1 can be also seen as particular cases in which we take $n = 1$ inside the dynamic questioning modality $[\sigma(n)?]$. We lift the restriction to single questions from the vanilla version of the language and allow questioning sequences in two stages. First by introducing special reduction axioms for minimal sequences, i.e., sequences of length two, in Section 5. Second, we introduce questioning sequences of arbitrary length and give fully general reduction axioms for them in Sections 4 and 6. We continue our exposition using the simplest version of IEL and afterwards return in Section 4 to considering the SQL extension in more detail.

3 The Tableau Synthesis Framework Applied to IEL

In order to obtain a sound, complete and terminating tableau calculus for IEL we apply the tableau synthesis framework introduced in [11, 10]. In brief, the tableau synthesis method works as follows. The user defines the formal semantics of their logic in the first-order specification language of the framework. The semantic specification can then be automatically transformed into tableau rules that form a calculus which is sound and complete provided the semantic specification satisfies certain conditions. In a next step the possibility to refine the tableau calculus in two ways is explored. First, it may be possible to refine that tableau rules by reducing their branching factor and, second, it may be possible to internalise semantic constructs of the tableau language in the language of the logic. Finally, the unrestricted blocking mechanism

can be added to the obtained calculus. The blocking mechanism ensures termination of the calculus if the logic has the finite model property. The final calculus is sound, complete and terminating, and, hence, provides the basis for a decision procedure for the logic.

The *object language* of specification of syntax of the logic IEL has several distinct sorts. The main sort of the language is the sort of formulae (sort f) which are denoted as $\varphi, \psi, \ldots$. Other sorts are individuals (sort i) denoted $i, j, \ldots$, propositional atoms (sort p) denoted $p, q, \ldots$, and agent labels (sort a) denoted as $a, a_0, a_1, \ldots$.

For reasons of economy and simplicity, we fix a (minimal) set of connectives for the syntax specification of IEL. The connectives and their types are listed in Figure 1.

Useful additions to the specification language include the *singleton set* operator $\{\cdot\}$ and the operator $\#\cdot$, which respectively link the individual and formula sorts, and the proposition and formula sorts. The operator $@_\cdot\cdot$ is the *at* (or *satisfaction*) operator, which is useful for internalising the semantic specification.

The *meta-language* of IEL for the specification of the semantics extends the object language of IEL with an additional domain sort d and the following symbols: binary predicate symbols (of type (d, d)) $R_{\overset{a}{\approx}}$, $R_{\overset{a}{\approx} \cap \overset{a}{\sim}}$, and $R_{\overset{a}{\sim}}$; the equality symbol $\dot{\approx}$ (we use a dot to distinguish equality from the issue relation); domain variables $x, y, z, \ldots$; and the first-order quantifiers $\forall$ and $\exists$. Finally, the meta-language contains three interpretation symbols ν_{i}, ν_{f}, and ν_{p}. For every nominal i of sort i, $\nu_{\mathsf{i}}(i)$ is a term of sort d. For every IEL-formula φ of sort f, proposition p of sort p, and term t of sort d, $\nu_{\mathsf{f}}(\varphi, t)$ and $\nu_{\mathsf{p}}(p, t)$ are atomic formulae in the semantic specification language for IEL.

Figure 2 shows the definition of the semantics of the IEL connectives in the meta-language. We give the standard Boolean and modal semantic definitions in the right column and the semantics of the sort-bridging connectives in the left column. Both are followed by definitions for the dynamic modalities. We denote the set of all these formulae by S_0.

Additionally there are conditions which specify properties of relations and equality. They are captured by the background theory axioms S^b which are listed in Figures 3 and 4.

The described semantic specification captures in first-order sentences the semantic conditions for IEL from Section 2. In particular, the difference between the semantic definition of the static modalities and the dynamic modalities becomes more apparent. While the static modalities are dropped by the definitions, the dynamic ones are only dropped in the definitions for atomic components in their scope. However, for complex formulae the dynamic modality is applied in the right hand side of the definition to a formula with lower complexity. This is also reflected in the semantic specifications that the reduction axioms vary depending on whether they are for propositional atoms, for singletons, and for formulae.

The next step is to transform the semantic specification into a normalised implicational form [11]. This is done by decomposing each logical equivalence of the specification S_0 into the left-to-right implication and the contrapositive of the right-to-left implication. The resulting sets of formulae are denoted by S^+ and S^-.

Connective	Type
$\{\cdot\}$	$\mathsf{i} \mapsto \mathsf{f}$
$@_\cdot\cdot$	$(\mathsf{i}, \mathsf{f}) \mapsto \mathsf{f}$
$\neg\cdot$	$\mathsf{f} \mapsto \mathsf{f}$
$\cdot \wedge \cdot$	$(\mathsf{f}, \mathsf{f}) \mapsto \mathsf{f}$

Connective	Type
$\#\cdot$	$\mathsf{p} \mapsto \mathsf{f}$
$Q_\cdot\cdot$	$(\mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$
$K_\cdot\cdot$	$(\mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$
$X_\cdot\cdot$	$(\mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$

Connective	Type
$[\cdot?]\cdot$	$(\mathsf{f}, \mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$
$[!]\cdot$	$(\mathsf{a}, \mathsf{f}) \mapsto \mathsf{f}$

Figure 1: Connectives of the object language of IEL.

$$\forall x(\nu_{\mathsf{f}}(\{i\},x) \leftrightarrow x \mathrel{\dot{\approx}} \nu_{\mathsf{f}}(i)) \qquad \forall x(\nu_{\mathsf{f}}(\neg\varphi,x) \leftrightarrow \neg\nu_{\mathsf{f}}(\varphi,x))$$
$$\forall x(\nu_{\mathsf{f}}(\#p,x) \leftrightarrow \nu_{\mathsf{p}}(p,x)) \qquad \forall x(\nu_{\mathsf{f}}(\varphi\wedge\psi,x) \leftrightarrow \nu_{\mathsf{f}}(\varphi,x)\wedge\nu_{\mathsf{f}}(\psi,x)))$$
$$\forall x(\nu_{\mathsf{f}}(@_i\varphi,x) \leftrightarrow \nu_{\mathsf{f}}(\varphi,\nu_{\mathsf{f}}(i))) \qquad \forall x(\nu_{\mathsf{f}}(Q_a\varphi,x) \leftrightarrow \forall y(R_{\mathrel{\underset{\approx}{\sim}}a}(x,y)\to\nu_{\mathsf{f}}(\varphi,y)))$$
$$\forall x(\nu_{\mathsf{f}}([\varphi?]_a\#p,x) \leftrightarrow \nu_{\mathsf{f}}(\#p,x))) \qquad \forall x(\nu_{\mathsf{f}}(K_a\varphi,x) \leftrightarrow \forall y(R_{\mathrel{\underset{\sim}{\sim}}a}(x,y)\to\nu_{\mathsf{f}}(\varphi,y)))$$
$$\forall x(\nu_{\mathsf{f}}([q]_a\{i\},x) \leftrightarrow \nu_{\mathsf{f}}(\{i\},x))) \qquad \forall x(\nu_{\mathsf{f}}(X_a\varphi,x) \leftrightarrow \forall y(R_{\mathrel{\underset{\approx}{\sim}}\cap\mathrel{\underset{\sim}{\sim}}a}(x,y)\to\nu_{\mathsf{f}}(\varphi,y))))$$
$$\forall x(\nu_{\mathsf{f}}([q]_a\neg\psi,x) \leftrightarrow \nu_{\mathsf{f}}(\neg[q]_a\psi,x))) \qquad \forall x(\nu_{\mathsf{f}}([q]_a(\psi\wedge\chi),x) \leftrightarrow \nu_{\mathsf{f}}([q]_a\psi,x)\wedge\nu_{\mathsf{f}}([q]_a\chi,x))$$
$$\forall x(\nu_{\mathsf{f}}([\varphi?]_aK_a\psi,x) \leftrightarrow \nu_{\mathsf{f}}(K_a[\varphi?]\psi,x)) \qquad \forall x(\nu_{\mathsf{f}}([!]_aQ_a\psi,x) \leftrightarrow \nu_{\mathsf{f}}(Q_a[!]_a\psi,x))$$
$$\forall x(\nu_{\mathsf{f}}([!]_aK_a\psi,x) \leftrightarrow \nu_{\mathsf{f}}(X_a[!]_a\psi,x)) \qquad \forall x(\nu_{\mathsf{f}}([!]_aX_a\psi,x) \leftrightarrow \nu_{\mathsf{f}}(X_a[!]_a\psi,x))$$
$$\forall x(\nu_{\mathsf{f}}([\varphi?]_aQ_a\psi,x) \leftrightarrow (\nu_{\mathsf{f}}(\varphi\wedge Q_a(\neg\varphi\vee[\varphi?]_a\psi),x)\vee\nu_{\mathsf{f}}(\neg\varphi\wedge Q_a(\varphi\vee[\varphi?]_a\psi),x))$$
$$\forall x(\nu_{\mathsf{f}}([\varphi?]_aX_a\psi,x) \leftrightarrow (\nu_{\mathsf{f}}(\varphi\wedge X_a(\neg\varphi\vee[\varphi?]_a\psi),x)\vee\nu_{\mathsf{f}}(\neg\varphi\wedge X_a(\varphi\vee[\varphi?]_a\psi),x))$$

Figure 2: Semantic specification S_0 of connectives for IEL ($q\in[\varphi?],[!]$)

$$\forall x\forall y(R_{\mathrel{\underset{\approx}{\sim}}\cap\mathrel{\underset{\sim}{\sim}}a}(x,y)\leftrightarrow R_{\mathrel{\underset{\approx}{\sim}}a}(x,y)\wedge R_{\mathrel{\underset{\sim}{\sim}}a}(x,y)),$$
$$\forall x\forall y\forall z((R_{\mathrel{\underset{\approx}{\sim}}a}(x,y)\wedge R_{\mathrel{\underset{\approx}{\sim}}a}(y,z))\to R_{\mathrel{\underset{\approx}{\sim}}a}(x,z)),\quad \forall x\forall y\forall z((R_{\mathrel{\underset{\sim}{\sim}}a}(x,y)\wedge R_{\mathrel{\underset{\sim}{\sim}}a}(y,z))\to R_{\mathrel{\underset{\sim}{\sim}}a}(x,z)),$$
$$\forall x\forall y\forall z((R_{\mathrel{\underset{\approx}{\sim}}\cap\mathrel{\underset{\sim}{\sim}}a}(x,y)\wedge R_{\mathrel{\underset{\approx}{\sim}}\cap\mathrel{\underset{\sim}{\sim}}a}(y,z))\to R_{\mathrel{\underset{\approx}{\sim}}\cap\mathrel{\underset{\sim}{\sim}}a}(x,z)),$$
$$\forall x\forall y(R_{\mathrel{\underset{\approx}{\sim}}a}(x,y)\to R_{\mathrel{\underset{\approx}{\sim}}a}(y,x)),\quad \forall x\forall y(R_{\mathrel{\underset{\approx}{\sim}}\cap\mathrel{\underset{\sim}{\sim}}a}(x,y)\to R_{\mathrel{\underset{\approx}{\sim}}\cap\mathrel{\underset{\sim}{\sim}}a}(y,x)),$$
$$\forall x\forall y(R_{\mathrel{\underset{\sim}{\sim}}a}(x,y)\to R_{\mathrel{\underset{\sim}{\sim}}a}(y,x)),\quad \forall xR_{\mathrel{\underset{\approx}{\sim}}a}(x,x),\quad \forall xR_{\mathrel{\underset{\approx}{\sim}}\cap\mathrel{\underset{\sim}{\sim}}a}(x,x),\forall xR_{\mathrel{\underset{\sim}{\sim}}a}(x,x)$$

Figure 3: Semantic specification of background theory axioms for the relations

It is not difficult to check that the semantic specification is well-defined in the sense of [11]. A semantic specification S is well defined iff S is normalized and the following conditions hold:

(wd1) $\forall S^0,\forall S^b \models \forall S$,

(wd2) the relation $\prec$ induced by S is a well-founded ordering on formulae, and

(wd3) for every formula $\varphi=\sigma(\varphi_1,\ldots,\varphi_m)$, defining a connective σ:

$$\forall S^0,\forall S^b \restriction \mathsf{sub}_\prec(\varphi) \models_c \forall\overline{x}\big(\big(\bigwedge\Phi^\varphi_+\to\phi^\sigma(\varphi_1,\ldots,\varphi_m,\overline{x})\big)\wedge\big(\phi^\sigma(\varphi_1,\ldots,\varphi_m,\overline{x})\to\bigvee\Phi^\varphi_-\big)\big)$$

Here Φ^φ_+ is the set obtained by collecting all instantiations of consequents from S^+ (the positive specifications in S) matching the formula φ. Φ^φ_- is the set obtained by collecting all instantiations of antecedents from S^- (the negative specifications in S) matching formula φ.

Condition (wd1) expresses the decomposition of the specification S to S^0 and S^b after normalization. The set S^0 contains the connective definitions, and the set S^b contains the background theory conditions. The set S^0 is further decomposed in two disjoint sets S^+ and S^-. Since the semantic specification is the union of the connective definitions and the background

$$\forall x(x\mathrel{\dot{\approx}}x),\quad \forall x\forall y(x\mathrel{\dot{\approx}}y\to y\mathrel{\dot{\approx}}x),\quad \forall x\forall y\forall z(x\mathrel{\dot{\approx}}y\wedge y\mathrel{\dot{\approx}}z\to x\mathrel{\dot{\approx}}z),$$
$$\forall\overline{p}\forall\overline{x}\forall y_i(x_i\mathrel{\dot{\approx}}y_i\to f(\overline{p},\overline{x})\mathrel{\dot{\approx}}f(\overline{p},x_1,\ldots,x_{i-1},y_i,x_{i+1},\ldots,x_n)),$$
$$\forall p\forall x\forall y(\nu_{\mathsf{f}}(\#p,x)\wedge x\mathrel{\dot{\approx}}y\to\nu_{\mathsf{f}}(\#p,y)),\quad \forall p\forall x\forall y(\nu_{\mathsf{p}}(p,x)\wedge x\mathrel{\dot{\approx}}y\to\nu_{\mathsf{p}}(p,y)).$$

Figure 4: Semantic specification of equality axioms

Tableau Expansion Rules (generated from $S_0 = S^+ \cup S^-$):

$$\frac{@_l[\varphi?]\Box_a\psi}{@_l\varphi,\ @_l\Box_a(\neg\varphi \vee [\varphi?]\psi) \mid @_l\neg\varphi,\ @_l\Box_a(\varphi \vee [\varphi?]\psi)}\ (\Box \in Q, X), \qquad \frac{@_l[!]K_a\psi}{@_l X_a[!]\psi}, \tag{3a}$$

$$\frac{@_l\neg[\varphi?]\Box_a\psi}{@_l(\neg\varphi \vee \neg\Box_a(\neg\varphi \vee [\varphi?]\psi)),\ @_l(\varphi \vee \neg\Box_a(\varphi \vee [\varphi?]\psi))}\ (\Box \in Q, X), \qquad \frac{@_l\neg[!]K_a\psi}{@_l\neg X_a[!]\psi}. \tag{3b}$$

$$\frac{@_l\blacksquare b}{@_l b}, \quad \frac{@_l\blacksquare\neg\varphi}{@_l\neg\blacksquare\varphi}, \quad \frac{@_l\blacksquare(\varphi\wedge\psi)}{@_l\blacksquare\varphi,\ @_l\blacksquare\psi}, \quad \frac{@_l[\varphi?]K_a\psi}{@_l K_a[\varphi?]\psi}, \quad \frac{@_l[!]\Box_a\varphi}{@_l\Box_a[!]\varphi}\ (\Box \in Q, X), \tag{3c}$$

$$\frac{@_l\neg\blacksquare b}{@_l\neg b}, \quad \frac{@_l\neg\blacksquare\neg\varphi}{@_l\blacksquare\varphi}, \quad \frac{@_l\neg\blacksquare(\varphi\wedge\psi)}{@_l\neg\blacksquare\varphi \mid @_l\neg\blacksquare\psi}, \quad \frac{@_l\neg[\varphi?]K_a\psi}{@_l\neg K_a[\varphi?]\psi}, \quad \frac{@_l\neg[!]\Box_a\varphi}{@_l\neg\Box_a[!]\varphi}\ (\Box \in Q, X), \tag{3d}$$

$$\frac{@_l\neg\Box_a\varphi}{@_l\Diamond_a\{f_{\neg\Box}(l,a,\varphi)\},\ @_{f_{\neg\Box}(l,a,\varphi)}\neg\varphi}\ (\Box \in Q, K, X), \qquad \frac{@_l\Box_a\varphi,\ @_l\Diamond_a\{l_2\}}{@_{l_2}\varphi}\ (\Box \in Q, K, X), \tag{3e}$$

$$\frac{@_l\neg\neg\varphi}{@_l\varphi}, \quad \frac{@_l\varphi\wedge\psi}{@_l\varphi,\ @_l\psi}, \quad \frac{@_l\neg(\varphi\wedge\psi)}{@_l\neg\varphi \mid @_l\neg\psi}, \tag{3f}$$

Background Theory Rules (generated from S_b):

$$\frac{@_l\widehat{X}_a\{l_2\}}{@_l\widehat{Q}_a\{l_2\},\ @_l\widehat{K}_a\{l_2\}}, \qquad \frac{@_l\widehat{Q}_a\{l_2\},\ @_l\widehat{K}_a\{l_2\}}{@_l\widehat{X}_a\{l_2\}}, \tag{3g}$$

$$\frac{@_l\Diamond_a\{l_2\}, @_{l_2}\{l_3\}}{@_l\Diamond_a\{l_3\}}, \quad \frac{@_l\{l\}}{@_l\Diamond_a\{l\}}, \quad \frac{@_l\Diamond_a\{l_2\}}{@_{l_2}\Diamond_a\{l\}}, \quad \frac{@_l\Diamond_a\{l_2\},\ @_{l_2}\Diamond_a\{l_3\}}{@_l\Diamond_a\{l_3\}}, \tag{3h}$$

$$\frac{@_l\Diamond_a\{l_2\}}{@_{l_2}\{l_2\}}, \frac{@_l\{l_2\}}{@_{l_2}\{l\}}, \frac{@_l\neg\{l_2\}}{@_{l_2}\neg\{l\}}, \quad \frac{@_l\varphi}{@_l\{l\}}, \frac{@_l\varphi, @_l\{l_2\}}{@_{l_2}\varphi}, \quad \text{(Clash):}\ \frac{@_l\varphi,\ @_l\neg\varphi}{\bot}. \tag{3i}$$

Figure 5: Refined calculus for IEL, where $\blacksquare \in \{[\varphi?]_a, [!]_a\}$, $b \in \{\#p, \{n\}\}$, $\Diamond \in \{\widehat{Q}, \widehat{K}, \widehat{X}\}$

theory, conditions (wd1) and (wd3) are trivially satisfied. Showing condition (wd2), i.e., well-foundedness of the ordering $\prec$ induced by the normalised specification, is more involved than usual because of the statements capturing the reduction axioms. Well-foundedness of the order can be established by assigning IEL formulae the following complexity measure:

$$c(p) = 1, \quad c(!) = 1, \quad c(\neg\varphi) = 1 + c(\varphi), \quad c(\varphi\wedge\psi) = 1 + \max(c(\varphi), c(\psi)), \tag{2a}$$

$$c(\Box_a\varphi) = 1 + c(\varphi) \quad \text{for } \Box_a \in \{K_a, Q_a, X_a\}, \text{ and} \tag{2b}$$

$$c([q]\psi) = (c(q) + 5)\cdot c(\psi) \quad \text{for } q \in \{\varphi?, !\}. \tag{2c}$$

Turning the normalised semantic specification into tableau rules in accordance with [11] then produces a sound and complete tableau calculus for checking satisfiability for IEL. We do not present this calculus here, but present (in Figure 5) immediately the calculus obtained after refinement.

Two refinements described in [11] have been applied. The first refinement is the internalisation of the domain symbols including interpretation symbols ν_{i}, ν_{f}, and ν_{p} in the language of the logic. For example, the rules generated for the $\Box$ operators are ($\Box \in \{Q, K, X\}$):

$$\frac{\nu_{\mathsf{f}}(\neg\Box_a\varphi, l)}{R(l, f_{\neg\Box}(l,a,\varphi)),\ \nu_{\mathsf{f}}(\neg\varphi, f_{\neg\Box}(l,a,\varphi))} \quad \text{and} \quad \frac{\nu_{\mathsf{f}}(\Box_a\varphi, l),\ l_2 \mathrel{\dot{\approx}} l_2}{\neg R(l, l_2) \mid \nu_{\mathsf{f}}(\varphi, l_2)}$$

R denotes the appropriate accessibility relation associated with $\Box_a$. $f_{\neg\Box}$ represents one of three fixed Skolem functions used as a convenient way to create witnesses for formulae of existential extent. Because IEL is a hybrid logic it fully supports individuals and the rules can be rewritten as

$$\frac{@_l\neg\Box_a\varphi}{@_l\Diamond_a\{f_{\neg\Box}(l,a,\varphi)\},\ @_{f_{\neg\Box}(l,a,\varphi)}\neg\varphi} \quad \text{and} \quad \frac{@_l\Box_a\varphi,\ @_{l_a}\Diamond_a\{l_2\}}{@_l\neg\Diamond_a\{l_2\} \mid @_{l_2}\varphi}.$$

Similarly for the other rules.

The second refinement attempts to replace branching rules by rules with fewer or no branches. For example, the rule for positive occurrences of $\Box$,

$$\frac{@_l\Box_a\varphi,\ @_{l_a}\Diamond_a\{l_2\}}{@_l\neg\Diamond_a\{l_2\} \mid @_{l_2}\varphi}, \quad \text{is refined to} \quad \frac{@_l\Box_a\varphi,\ @_l\Diamond_a\{l_2\}}{@_{l_2}\varphi}.$$

Other refined rules in the presented calculus are the rules expressing triangular properties in (3g) and (3h). These rule refinements are justified because the (†) condition from [11] can be shown to hold in each case. The presented calculus is therefore sound and complete for IEL.

Finally, if the logic has the finite model property then the generated tableau calculus can be turned into a decision procedure by adding the blocking mechanism introduced in [9] which is based on the following unrestricted blocking rule:

$$\text{(UB):}\frac{@_l\{l\},\ @_{l_0}\{l_0\}}{@_l\{l_0\} \mid @_l\neg\{l_0\}}$$

For the static part of IEL the finite model property is obtained by a standard filtration argument. The reduction axioms introduced before provide a way to translate formulae from the dynamic part to equivalent formulae in the static language. The translation is:

$$t(p) = p, \quad t(\neg\varphi) = \neg t(\varphi), \quad t(\varphi\wedge\psi) = t(\varphi)\wedge t(\psi), \tag{4a}$$
$$t(\Box_a\varphi) = \Box_a t(\varphi) \quad \text{for } \Box_a \in \{K_a, Q_a, X_a\}, \tag{4b}$$
$$t(lhs) = t(rhs) \quad \text{for the reduction axioms in (1a)–(1d).} \tag{4c}$$

This implies that IEL has the finite model property and we can obtain the following result:

Theorem 1. *The calculus listed on Figure 5 is sound and complete for* IEL *satisfiability and it is also terminating if equipped with the unrestricted blocking mechanism.*

4 Extension to Sequential Questioning Logic

In this section we generalize the IEL/DELQ framework to a setting using questioning sequences and we call the emerging theory *Sequential Questioning Logic* (henceforth, *SQL*).

The language of SQL is recursively defined by the following BNF:

$$\varphi \quad ::= \quad n \mid p \mid \neg\varphi \mid \varphi\vee\varphi \mid \Box\varphi \mid [\sigma(k)?]\varphi \mid [!]\varphi$$

with $n, p, a, \neg, \vee, \Box, [!]$ as before and $\sigma(n)?$ representing dynamic questioning actions where $\sigma(n) = \langle\varphi_0, \ldots, \varphi_{n-1}\rangle$ is a sequence of SQL formulae. The semantics of the operators is also as before with the generalized intersection already introduced in Section 2 used for sequences. The fact that the questioning modalities are the only ones different between IEL and SQL dialects allows us to add questioning sequences while preserving all the other components.

Figure 6: SQL specific dynamic connectives, semantics and tableau rules

conn.	type	semantics	rules
$[\cdot?]$	$[\mathsf{f}] \mapsto \mathsf{f}$	$\forall x(\nu_{\mathsf{f}}([\sigma?]\#p, x) \leftrightarrow \nu_{\mathsf{f}}(\#p, x)))$ $\forall x(\nu_{\mathsf{f}}([\sigma?]\{n\}, x) \leftrightarrow \nu_{\mathsf{f}}(\{n\}, x)))$ $\forall x(\nu_{\mathsf{f}}([\sigma?]\neg\varphi, x) \leftrightarrow \nu_{\mathsf{f}}(\neg[\sigma?]\varphi, x)))$ $\forall x(\nu_{\mathsf{f}}([\sigma?](\varphi \wedge \psi), x) \leftrightarrow \nu_{\mathsf{f}}([\sigma?]\varphi, x) \wedge \nu_{\mathsf{f}}([\sigma?]\psi, x))$ $\forall x(\nu_{\mathsf{f}}([\sigma?]K_a\varphi, x) \leftrightarrow \nu_{\mathsf{f}}(K_a[\sigma?]\varphi, x))$	As in Figure 5 with $[\sigma?]$ instead of $[\varphi?]$
		$\forall x(\nu_{\mathsf{f}}([\sigma(n)?]Q_a\varphi, x) \leftrightarrow$ see below $\forall x(\nu_{\mathsf{f}}([\sigma(n)?]X_a\varphi, x) \leftrightarrow$ see below	See the generalized rules below

The static part of SQL brings nothing new, it is axiomatized, as before in IEL, by standard hybrid logic axioms for nominals and intersection. The dynamic part of SQL brings some new features that generalize the initial setting from IEL, and we do not require formulae in $\sigma(n)$ to induce a partition of the domain.

The significant new feature in SQL relative to IEL is the presence of dynamic questioning modalities over sequences of questions. This has to bring about new reduction axioms. For formulae φ with factual content the jump to questioning sequences is an obvious generalization of the pattern in previous reduction axioms. For such φ_0, φ_1 in a minimal sequence we have:

$$\begin{aligned}[\varphi_0, \varphi_1]X\varphi \leftrightarrow\ & (\varphi_0 \wedge \varphi_1 \wedge X((\varphi_0 \wedge \varphi_1) \to [\varphi_0, \varphi_1]\varphi)) \\ \vee\ & (\varphi_0 \wedge \neg\varphi_1 \wedge X((\varphi_0 \wedge \neg\varphi_1) \to [\varphi_0, \varphi_1]\varphi)) \\ \vee\ & (\neg\varphi_0 \wedge \varphi_1 \wedge X((\neg\varphi_0 \wedge \varphi_1) \to [\varphi_0, \varphi_1]\varphi)) \\ \vee\ & (\neg\varphi_0 \wedge \neg\varphi_1 \wedge X((\neg\varphi_0 \wedge \neg\varphi_1) \to [\varphi_0, \varphi_1]\varphi)).\end{aligned}$$

This generalizes in the expected way to longer factual sequences. However, this cannot be extended beyond factual formulae, not even for minimal questioning sequences of length two. This approach fails for complex formulae that have questioning content or, in general, extra-factual or higher-order content. Consider as an illustration the following complex questioning formula: $\xi := (\widehat{Q}i \to (j \vee k)) \wedge ((\widehat{Q}j \wedge p) \to \widehat{Q}i)$. A model with a domain of three worlds i, j, k, universal issue and epistemic relations, and a valuation that makes p true at k provides a counterexample when we make the following substitutions: $\varphi_0 \mapsto \xi,\ \varphi_1 \mapsto \xi,\ \varphi \mapsto \neg p$.

For questioning sequences the disjunctive structure of the reduction axiom has to induce a partition of the domain. However, the questioning sequence does not have to give rise to a partition. This difference is not always fully understood and appreciated. If the questioning sequence has a more complex structure, for instance, if it induces a cover of the domain, more complex patterns are needed in the reduction axiom, that can ensure that the right hand side remains an exhaustive exclusive disjunction. In this way, fully general reduction axioms for SQL can be obtained and they will have the pattern given below.

We present the new additions in a synthetic way in Table 6. The questioning modalities have a different type than before as they are now defined over lists of formulae. Except for this

type difference most of the tableau rules will be as before. The new connectives using sequences will have new reduction axioms and new rules as specified in the table.

The reduction axioms will have the following pattern, for $\blacksquare \in \{Q, X\}$:

$$[\sigma(n)]\blacksquare_a\varphi \leftrightarrow \bigvee_{i=0}^{2^{|\sigma(n)|}} \Big(\bigwedge_{k=0}^{|\sigma(n)|} ([\sigma(k-1)]\varphi_k)^{\beta(i)(k)} \wedge \blacksquare_a\big(\bigwedge_{k=0}^{|\sigma(n)|} ([\sigma(k-1)]\varphi_k)^{\beta(i)(k)} \rightarrow [\sigma(n)]\varphi\big)\Big),$$

where $|\sigma(n)|$ is the length of the questioning sequence $\sigma(n) = \langle \varphi_0, .., \varphi_{n-1} \rangle$,

$$\text{and the value of } \varphi^{\beta(k)(i)} \text{ is determined by: } \varphi^{\beta(k)(i)} = \begin{cases} \varphi & \text{if } \beta(k)(i) = 1, \text{ and} \\ \neg\varphi & \text{if } \beta(k)(i) = 0. \end{cases}$$

$\beta(k)(i)$ represents the i-th position in the binary encoding $\beta(k)$ of the decimal number k.

The corresponding tableau rules are obtained as follows, for $\chi_i^k = \bigwedge_{k=0}^{|\sigma(n)|}([\sigma(k-1)]\varphi_k)^{\beta(i)(k)}$:

$$\frac{@_l[\sigma?]\blacksquare_a\varphi}{@_l \bigwedge_{k=0}^{|\sigma(n)|} \chi_1^k \wedge \blacksquare_a(\bigwedge_{k=0}^{|\sigma(n)|} \chi_1^k \rightarrow [\sigma(n)]\varphi) \mid \cdots \mid @_l \bigwedge_{k=0}^{|\sigma(n)|} \chi_n^k \wedge \blacksquare_a(\bigwedge_{k=0}^{|\sigma(n)|} \chi_n^k \rightarrow [\sigma(n)]\varphi)}$$

In addition the complexity function for formulae has to add to Equation 2 values that take into account the length of the questioning sequences.

5 Implementing an IEL Prover with MetTeL^2

In this section, we describe our experience in using MetTeL^2 [13] to generate a tableau prover for the tableau calculus derived in the previous section. MetTeL^2 is a prototypical tableau prover generator developed with the tableau synthesis framework as its theoretical foundation. Given the specification of a logic and the specification of a tableau calculus for this logic MetTeL^2 generates Java code for a tableau prover implementing the tableau calculus. MetTeL^2 has been successfully applied to several of logics, including Boolean logic, modal logics K, KT, S4, description logics $\mathcal{ALCO}$ and $\mathcal{ALBO}$id, and a hybrid logic with global counting operators [6]. The list is constantly growing. These test cases and downloadable copies of the generated provers are publicly available from the MetTeL website.

The underlying language for syntax specification of IEL in MetTeL^2 is in line with the tableau synthesis framework object specification language. As the object language is settled in Section 3, preparing the syntax specification for MetTeL^2 is straightforward. For example, the syntax specification contains declaration of four sorts.

```
sort formula, agent, prop, individual;
```

Further, declarations of each connectives follow their representation in Figure 1. For instance, the connective # is specified by the following declaration.

```
formula proposition = '#' prop;
```

The declaration of the dynamic modality for questioning is given by:

```
formula query = '[?' formula ']' agent formula;
```

The syntax for Skolem terms which are fresh labels introduced during the application of diamond rules is given as follows.

```
individual fq = 'fq' '(' individual ',' agent ',' formula ')';
individual fk = 'fk' '(' individual ',' agent ',' formula ')';
individual fx = 'fx' '(' individual ',' agent ',' formula ')';
```

The specification of the tableau calculus in MetTeI^2 reflects all the rules of Figure 5. There are decomposition rules for positive as well as negative occurrences of all connectives. The exception is negation, which only has a rule for negative occurrence, namely elimination of double negation. For example, the rules for the three static modalities are specified as follows.

```
@l<q> A P / @l <q> A {fq(l,A,P)} @fq(l,A,P)P priority 7$;
@l<k> A P / @l <k> A {fk(l,A,P)} @fk(l,A,P)P priority 7$;
@l<x> A P / @l <x> A {fx(l,A,P)} @fx(l,A,P)P priority 7$;
@l ~(<q> A P) @l <q> A {l2} / @l2~P priority 2$;
@l ~(<k> A P) @l <k> A {l2} / @l2~P priority 2$;
@l ~(<x> A P) @l <x> A {l2} / @l2~P priority 2$;
```

The rules for dynamic modalities have specific cases for atomic formulae, i.e., propositional atoms or nominals, and cases for complex formulae with non-factual content: questioning, epistemic or both. The cases for atomic formulae are specified as follows.

```
@l ([?P] A #B) / @l #B priority 2$;
@l ([?P] A {l2}) / @l ({l2}) priority 2$;
@l ([!] A {l2}) / @l {l2} priority 2$;
@l ([!] A #B) / @l #B priority 2$;
@l ~([?P] A #B) / @l ~(#B) priority 2$;
@l ~([?P] A {l2}) / @l ~{l2} priority 2$;
@l ~([!] A {l2}) / @l ~{l2} priority 2$;
@l ~([!] A #B) / @l ~(#B) priority 2$;
```

Having the rules for intersection of accessibility relations in the background theory is important because it plays a crucial role in the reduction rule for the dynamic modalities:

```
@l<q> A {l2} @l<k> A {l2} / @l<x> A {l2} priority 2$;
@l<x> A {l2} / @l<q> A {l2} @l<k> A {l2} priority 2$;
```

In MetTeI^2, appearance of an equality formula in a branch immediately triggers ordered rewriting within the branch. The unrestricted blocking rule is implemented with use of this feature and the equality formula on its left branch forces the branch to be rewritten with respect to the additional equality.

An important feature provided by the MetTeI^2 implementation is the possibility to assign priorities to the tableau rules. This can be used to control the way the rules are applied in the generated prover. Each rule is followed by a number, which defines the rule application priority. Rules with smaller priority values have higher priority and applied more eagerly. Use of this feature is essential for the efficiency of the generated provers and especially in the case of IEL because the rules corresponding to reduction axioms with disjunctive patterns can be assigned lower priorities (higher priority values), thus reducing the branching factor and improving efficiency.

From the syntax specification and the tableau calculus, a tableau prover for IEL is automatically generated by MetTeI^2 according to the process described in detail in [13]. The generated prover can be used like most other tableau provers. Given a set of formulae as an input, the prover returns an answer **`Satisfiable`** or **`Unsatisfiable`** together with a model in the first case or with a set of contradictory formulae in the latter case.

We have tested the generated prover on a small set of sample formulae, where all the answers were correct.

6 Implementing IEL and SQL in `Haskell`

The second implementation uses the literate `Haskell` script `Qtab.lhs`. In this section, we provide a brief description of this implementation and include the implementation itself in the long version of the paper [8]. The implementation started from a pre-existing tableau prover for hybrid logic [16], to which we have added the details needed to model dynamic questioning actions. The tableau construction functionality was also completely changed. The current setting is more congenial with the framework from [11], which includes having a background theory for intersection and using unrestricted blocking.

We give next a broad view of the modules contained in the `Qtab.lhs` architecture and their functionality. `Syntax.hs`: Preliminary module containing the data structures for modal and first-order logic formulae as well as the tableaux. `Qtab.lhs`: The module containing the main functionality for the tableau prover such as the `decide` function that takes a formula in the IEL language and decides if it is or is not a tautology. Also functions controlling the order in which the formulae are analysed. `Decomp.hs`: The module containing the functionality associated with tableau expansion rules by logical decomposition. This proceeds either by standard logical analysis or by rules synthesized from reduction axioms. `Backgrd.hs`: The module containing the main components of the IEL background theory. In particular, the rules governing the behaviour of nominals and the rules for intersection are defined here. Also the unrestricted blocking mechanism is handled by functions in this module. `Divide.hs`: The order in which branches in a tableau are expanded is determined by their syntactic structure. The module contains functions used to recognize structural properties of formulae and to divide tableau nodes into component lists of prioritised formulae. `Auxilar.hs`: The module containing auxiliary functionality (such as displaying tableaux and translating formulae).

One particular aspect in which the `Haskell` implementation proved to be useful was in dealing with questioning sequences. Questioning sequences can also be added in MetTel2, see the rules for sequences of length two in the appendix of the long version. The features of functional programming and the way in which recursion is implicitly built in `Haskell` definitions makes working with arbitrary sequences of questions easier. It also made it obvious that modalities capturing questioning sequences can be modelled as fully functional algebraic data structures suitable for recursive manipulation. The decomposition rules for the static connectives and resolution are as in Figure 5. We include several illustrations of `Qtab.lhs` output for questioning sequences of length two in the long version. The decomposition rules for the general case of arbitrarily long sequences follow the pattern of the rules from Table 6. We include below some illustrative examples of prover output for some paradigmatic examples of SQL formulae:

```
*Sql> deciden (Quest [Prop (P 0), Prop (Q 0)] (Box 2 (Disj [Prop (P 0), Prop (Q 0)])))
(False,5)
*Sql> deciden (Quest [Prop (P 0), Prop (Q 0)] (Box 1 (Disj [Prop (P 0), Prop (Q 0)])))
(False,14)
*Sql> deciden (Quest [Prop (P 0)] (Reso  (Box 2 (Prop (P 0)))))
(False,12)
```

The first example illustrates a questioning sequence of length two combined with the static knowledge modality, which has a commutating behaviour. The second example illustrates a questioning sequence of length two in combination with the static issue modality, which uses reduction axioms based on the disjunction pattern: The third example illustrates a questioning sequence combining both asking actions and resolution actions. Because the resolution modality is idempotent, all resolution sequences are equivalent to a sequence of length one. Therefore the following reduction axioms are used for dealing with the aspect of the interaction between

a questioning-sequence followed by a resolution-sequence:

$$[\sigma?][!]Q_a\psi \leftrightarrow [\sigma?]Q_a[!]\psi, \quad [\sigma?][!]K_a\psi \leftrightarrow [\sigma?]X_a[!]\psi, \quad [\sigma?][!]X_a\psi \leftrightarrow [\sigma?]X_a[!]\psi \tag{5a}$$

The final illustration shows the difference between knowing that and knowing whether. After a yes/no question about p the issue relation decides whether p, this turns out to be a SQL validity, however, it does not settle that p holds.

```
*Sql> (deciden (Neg (Quest [Prop (P 0), Neg (Prop (P 0))] (Box 1 (Prop (P 0))))))
(False,40)

*Sql> (deciden (Neg (Quest [Prop (P 0), Neg (Prop (P 0))]
      (Disj [Box 1 (Prop (P 0)), Box 1 (Neg (Prop (P 0)))]))))
(True,84)
```

More detailed code output, traces of step-by step inference and tableau generation, and further explanation of the code functionality is included in the long version of the paper.

7 Concluding Remarks

In this paper we have shown what can be achieved when applying tableau synthesis and implementation for dynamic modalities of the simplest kind. This is only an initial illustration that serves as a case study for further extensions. We have considered one such extension to questioning sequences. Further extensions that we want to consider in the future include dynamic questioning actions that can model privacy and insecure communication and employ product update [2, 1] for computing issue relations [7] and a richer repertoire of questioning actions that go beyond the propositional case and include wh-questions [5, 18].

MetTeL2 provides a robust and efficient platform for automatically generating a tableau prover for IEL. On the small set of formulae we used for testing, MetTeL2 was faster than the `Haskell` prover, because it implements clever backtracking techniques and other optimisations, currently not supported in the `Haskell` implementation.

On the other hand, the `Haskell` implementation provides a framework in which more experimental features of further extensions can be easily programmed and tested before they are ready to become mainstream conditions. We used the case of SQL to illustrate such an extension. We conclude with two main points about the overall significance of our approach.

We have shown how a dynamic component, in particular, dynamic questioning actions, can be integrated in the tableau synthesis framework. Based on the synthesised tableau calculus, two implementations have been developed: MetTeL2 and `Qtab.lhs`. This dynamic extension relies on rules in which the complexity of the formulae inside the scope of the dynamic modalities is reduced, even if the complexity of the conclusion formula in the rule can increase.

The second contribution facilitated by the implementations is an extension of the underlying dynamic logic itself. Implementing the reduction details made it obvious that a logical language containing sequences of questions, not just modalities for questioning actions, can be modelled in the framework, and extends the dynamic logic in a useful direction.

References

[1] A. Baltag. Logics for insecure communication. In *Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 111–121. Morgan Kaufmann, 2001.

[2] A. Baltag, L. S. Moss, and S. Solecki. The logic of public announcements, common knowledge, and private suspicions. In *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '98, pages 43–56. Morgan Kaufmann, 1998.

[3] L. del Cerro, D. Fauthoux, O. Gasquet, A. Herzig, D. Longin, and F. Massacci. Lotrec: the generic tableau prover for modal and description logics. In *Proceedings of the First International Joint Conference on Automated Reasoning*, pages 453–458. Springer, 2001.

[4] H. Ditmarsch, W. Hoek, and B. Kooi. *Dynamic epistemic logic*. Springer, 2007.

[5] J. Hintikka, I. Halonen, and A. Mutanen. Interrogative logic as a general theory of reasoning. In D. M. Gabbay, R. H. Johnson, H. J. Ohlbach, and J. Woods, editors, *Handbook of logic of argument and inference: The turn towards the practical*, pages 295–337. Elsevier, 2002.

[6] M. Khodadadi, R. A. Schmidt, D. Tishkovsky, and M. Zawidzki. Terminating tableau calculi for modal logic K with global counting operators. Manuscript, `http://www.mettel-prover.org/papers/KEn12.pdf`, 2012.

[7] Ş. Minică. *Dynamic Logic of Questions*. PhD thesis, ILLC, University of Amsterdam, 2011.

[8] Ş. Minică, M. Khodadadi, D. Tishkovsky, and R. A. Schmidt. Synthesising and implementing tableau calculi for interrogative epistemic logics, 2012. Long version of the present paper, `http://www.mettel-prover.org/papers/IEL-long.pdf`.

[9] R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. In *Proc. ISWC 2007 + ASWC 2007*, volume 4825 of *LNCS*, pages 438–451. Springer, 2007.

[10] R. A. Schmidt and D. Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In *Proc. IJCAR 2008*, volume 5195 of *LNCS*, pages 194–209. Springer, 2008.

[11] R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Logical Methods in Computer Science*, 7(2):1–32, 2011.

[12] B. D. ten Cate. *Model Theory for Extended Modal Languages*. PhD thesis, ILLC, University of Amsterdam, 2005.

[13] D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. MetTeL2: Towards a tableau prover generation platform. In these Proceedings, 2012.

[14] J. van Benthem. *Logical dynamics of information and interaction*. Cambridge Univ. Press, 2011.

[15] J. van Benthem and Ş. Minică. Toward a dynamic logic of questions. In *Logic, Rationality, and Interaction*, volume 5834 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2009.

[16] J. van Eijck. Hylotab: Tableau-based theorem proving for hybrid logics, 2002. Manuscript, CWI, Amsterdam.

[17] J. van Eijck. DEMO: A demo of epistemic modelling. In *Interactive Logic. Selected Papers from the 7th Augustus de Morgan Workshop, London*, pages 303–362, 2007.

[18] A. Wiśniewski. Erotetic search scenarios, problem solving, and deduction. *Logique & Analyse*, 185-188:139–166, 2004.

A Appendix: Implementation Details and Illustrations

A.1 MetTeL^2 Specification of IEL

The specification of the syntax of IEL for input to MetTeL^2 is the following:

```
specification IEL;

syntax IEL{
    sort formula, individual, prop, agent;

    formula true = 'true';
    formula false = 'false';
    formula singleton = '{' individual '}';
    formula atom = '#' prop;
    formula negation = '~' formula;
    formula diamondQ = '<q>' agent formula;
    formula diamondK = '<k>' agent formula;
    formula diamondX = '<x>' agent formula;
    formula at = '@' individual formula;

    formula query = '[?' formula ']' agent formula;
    formula resol = '[!]' agent formula;
    formula disjunction  = formula '|' formula;
    formula equality = '[' individual '=' individual ']';

    individual fq = 'fq' '(' individual ',' agent ',' formula ')';
    individual fk = 'fk' '(' individual ',' agent ',' formula ')';
    individual fx = 'fx' '(' individual ',' agent ',' formula ')';
}
```

A.2 MetTeL^2 Specification of the Tableau Calculus for IEL

The tableau rules of the IEL calculus for input to MetTeL^2 are given below:

```
//Equality rules
@l{l2} / @l2{l} priority 1$;
@l~{l2} / @l2{l2} priority 1$;
@l P / @l{l} priority 1$;
@l <q> A {l2} / @l2 {l2} priority 1$;
@l <k> A {l2} / @l2 {l2} priority 1$;
@l <x> A {l2} / @l2 {l2} priority 1$;
@l P  @l{l2} /  @l2 P priority 2$;
@l <x> A {l2}  @l2 {l3} / @l <x> A {l3} priority 2$;
@l <q> A {l2}  @l2 {l3} / @l <q> A {l3} priority 2$;
@l <k> A {l2}  @l2 {l3} / @l <k> A {l3} priority 2$;

//Decomposition rules
@l ~(~P) / @l P priority 1$;
@l(P|Q) / @l P $| @l Q priority 3$;
@l~(P|Q) / @l~P @l~Q priority 1$;
@l<q> A P / @l <q> A {fq(l,A,P)} @fq(l,A,P)P priority 7$;
@l<k> A P / @l <k> A {fk(l,A,P)} @fk(l,A,P)P priority 7$;
@l<x> A P / @l <x> A {fx(l,A,P)} @fx(l,A,P)P priority 7$;
@l ~(<q> A P) @l <q> A {l2} / @l2~P priority 2$;
@l ~(<k> A P) @l <k> A {l2} / @l2~P priority 2$;
@l ~(<x> A P) @l <x> A {l2} / @l2~P priority 2$;

```

```
@l ([?P] A #B) / @l #B priority 2$;
@l ~([?P] A #B) / @l ~(#B) priority 2$;
@l ([?P] A {l2}) / @l ({l2}) priority 2$;
@l ~([?P] A {l2}) / @l ~{l2} priority 2$;
@l ([!] A {l2}) / @l {l2} priority 2$;
@l ~([!] A {l2}) / @l ~{l2} priority 2$;
@l ([!] A #B) / @l #B priority 2$;
@l ~([!] A #B) / @l ~(#B) priority 2$;

@l ([?P] A ~Q) / @l ~([?P] A Q)$;
@l ([!] A ~Q) / @l ~([!] A Q)$;
@l ([?P] A (P1 | P2)) / @l ([?P] A P1) $| @l ([?P] A P2)$;
@l ([!] A (P1 | P2)) / @l ([!] A P1) $| @l ([!] A P2)$;
@l ([!] A ~(<q> A ~P)) / @l ~(<q> A ~([!] A P))$;
@l ([!] A ~(<x> A ~P)) / @l ~(<x> A ~([!] A P))$;
@l ([?P] A ~(<k> A ~Q)) / @l ~(<k> A ~([?P] A Q))$;
@l ([!] A ~(<k> A ~P)) / @l ~(<x> A ~([!] A P))$;
@l ([?P] A ~(<q> A ~Q)) / @l ~(~P | <q> A ~(~P | [?P] A Q)) $| @l ~(P | <q> A ~(P | [?P] A Q))$;
@l ([?P] A ~(<x> A ~Q)) / @l ~(~P | <x> A ~(~P | [?P] A Q)) $| @l ~(P | <x> A ~(P | [?P] A Q))$;

@l ~([?P] A ~Q) / @l ([?P] A Q)$;
@l ~([!] A ~Q) / @l ([!] A Q)$;
@l ~([?P] A (P1 | P2)) / @l ~([?P] A P1)  @l ~([?P] A P2)$;
@l ~([!] A (P1 | P2)) / @l ~([!] A P1)  @l ~([!] A P2)$;
@l ~([!] A ~(<q> A ~P)) / @l <q> A ~([!] A P)$;
@l ~([!] A ~(<x> A ~P)) / @l <x> A ~([!] A P)$;
@l ~([?P] A ~(<k> A ~Q)) / @l (<k> A ~([?P] A Q))$;
@l ~([!] A ~(<k> A ~P)) / @l (<x> A ~([!] A P))$;
@l ~([?P] A ~(<q> A ~Q)) / @l (~P | <q> A ~(~P | [?P] A Q)) @l (P | <q> A ~(P | [?P] A Q))$;
@l ~([?P] A ~(<x> A ~Q)) / @l (~P | <x> A ~(~P | [?P] A Q)) @l (P | <x> A ~(P | [?P] A Q))$;

// Theory rules
@l<q> A {l2} @l<k> A {l2} / @l<x> A {l2} priority 2$;
@l<x> A {l2} / @l<q> A {l2} @l<k> A {l2} priority 2$;
@l {l} / @l <q> A {l} priority 1$;
@l {l} / @l <k> A {l} priority 1$;
@l {l} / @l <x> A {l} priority 1$;
@l <q> A {l2} @l2 <q> A {l3} / @l <q> A {l3} priority 2$;
@l <q> A {l2} / @l2 <q> A {l} priority 3$;
@l <k> A {l2} @l2 <k> A {l3} / @l <k> A {l3} priority 2$;
@l <k> A {l2} / @l2 <k> A {l} priority 3$;
@l <x> A {l2} @l2 <x> A {l3} / @l <x> A {l3} priority 2$;
@l <x> A {l2} / @l2 <x> A {l} priority 3$;

// Closure rule
@l P  @l~P /  priority 0$;

//Blocking related
@l{l0} / [l=l0] priority 1$;
[l=l0] / @l{l0} priority 1$;
@l~{l0} / ~([l=l0]) priority 1$;
~([l=l0]) / @l~{l0} priority 1$;
@l{l} @l0{l0} / [l=l0] $| ~([l=l0]) priority 6$;
```

A.3 MetTeL^2 Extension for Handling Questioning Sequences

The calculus can be extended to questioning sequences by adding more rules to MetTeL^2. We include below the rules for SQL questioning sequences of length two in positive format:

```
// Dynamic Sequences (Length Two)
@1 ([Qp] A P1 ([Qp] A P2 {{B}})) / @1 {{B}} priority 2$;
@1 ([Qp] A P  ([Rp] A {{B}})) / @1 {{B}} priority 2$;
@1 ([Qn] A P1 ([Qn] A P2 {12})) / @1 {12} priority 2$;
@1 ([Qn] A P  ([Rn] A {12})) / @1 {12} priority 2$;
@1 ([Q] A P1 ([Q] A P2 ~Q)) / @1 ~([Q] A P1 ([Q] A P2 Q))$;
@1 ([Q] A P ([R] A ~Q)) / @1 ~([Q] A P ([R] A Q))$;
@1 ([Q] A P1 ([Q] A P2 (P3 & P4))) / @1 ([Q] A P1 ([Q] A P2 P3))  @1 ([Q] A P1 ([Q] A P2 P4))$;
@1 ([Q] A P ([R] A (P1 & P2))) / @1 ([Q] A P ([R] A P1))  @1 ([Q] A P ([R] A P2))$;
@1 ([Q] A P ([R] A [q] A P )) / @1 (P & [q] A (P -> [Q] A P ([R] A Q))) $|
                                @1 (~P & [q] A (~P -> [Q] A P ([R] A Q)))$;
@1 ([Q] A P ([R] A [x] A P )) / @1 (P & [x] A (P -> [Q] A P ([R] A Q))) $|
                                @1 (~P & [x] A (~P -> [Q] A P ([R] A Q)))$;
@1 ([Q] A P1 ([Q] A P2 [k] A Q)) / @1 ([k] A  ([Q] A P1 ([Q] A P2 Q)))$;
@1 ([Q] A P ([R] A [k] A Q)) / @1 (P & [x] A (P -> [Q] A P ([R] A Q))) $|
                               @1 (~P & [x] A (~P -> [Q] A P ([R] A Q)))$;
@1 ([Q] A P1 ([Q] A P2 [q] A Q)) / @1 (( P & [Q] A P1 P2)  & [q] A (( P & [Q] A P1 P2)  -> [Q] A P1 ([Q] A P2 Q))) $|
                                   @1 ((~P & [Q] A P1 P2)  & [q] A ((~P & [Q] A P1 P2)  -> [Q] A P1 ([Q] A P2 Q))) $|
                                 @1 (( P & ~([Q] A P1 P2)) & [q] A (( P & ~([Q] A P1 P2)) -> [Q] A P1 ([Q] A P2 Q))) $|
                                 @1 ((~P & ~([Q] A P1 P2)) & [q] A (( P & ~([Q] A P1 P2)) -> [Q] A P1 ([Q] A P2 Q))) $;
@1 ([Q] A P1 ([Q] A P2 [x] A Q)) / @1 (( P & [Q] A P1 P2)  & [x] A (( P & [Q] A P1 P2)  -> [Q] A P1 ([Q] A P2 Q))) $|
                                 @1 ((~P & [Q] A P1 P2)  & [x] A ((~P & [Q] A P1 P2)  -> [Q] A P1 ([Q] A P2 Q))) $|
                                 @1 (( P & ~([Q] A P1 P2)) & [x] A (( P & ~([Q] A P1 P2)) -> [Q] A P1 ([Q] A P2 Q))) $|
                                 @1 ((~P & ~([Q] A P1 P2)) & [x] A (( P & ~([Q] A P1 P2)) -> [Q] A P1 ([Q] A P2 Q))) $;
```

A.4 Sample Invocations of the Generated Prover

We include below output of two runs of the MetTeL2 generated prover. The examples illustrate the difference in expressive power between a logic with nominals and one without nominals. The intersection modality cannot be defined in a language without names for worlds.

```
//Give input for satisfiability check
@1 ~( ~(<q> a P) | ~( <k> a P) | (<x> a P) )

Satisfiable.
Model: [( ~ ( [ ( fq ( 1 , a , P ) ) = 1 ] ) ), ( @ ( fq ( 1 , a , P ) ) ( ~ ( { 1 } ) ) ),
( @ 1 ( { 1 } ) ), ( @ 1 ( ~ P ) ), ( @ 1 ( ~ ( ~ (  a P ) ) ) ), ( @ 1 ( ~ ( ~ (  a P ) ) ) ),
( @ 1 ( ~ (  a P ) ) ), ( @ 1 ( ~ ( ( ~ (  a P ) ) | ( ~ (  a P ) ) ) ) ),
( @ 1 ( ~ ( ( ( ~ (  a P ) ) | ( ~ (  a P ) ) ) | (  a P ) ) ) ), ( @ 1 (  A ( { 1 } ) ) ),
( @ 1 (  a P ) ), ( @ 1 (  a ( { 1 } ) ) ), ( @ 1 (  a ( { ( fq ( 1 , a , P ) ) } ) ) ),
( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  a P ) ), ( @ 1 (  a ( { 1 } ) ) ),
( @ 1 (  a ( { ( fk ( 1 , a , P ) ) } ) ) ), ( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  a ( { 1 } ) ) ),
( @ ( fq ( 1 , a , P ) ) P ), ( @ ( fq ( 1 , a , P ) ) ( { ( fq ( 1 , a , P ) ) } ) ),
( @ ( fq ( 1 , a , P ) ) (  A ( { ( fq ( 1 , a , P ) ) } ) ) ), ( @ ( fq ( 1 , a , P ) ) (  a ( { 1 } ) ) ),
( @ ( fq ( 1 , a , P ) ) (  a ( { ( fq ( 1 , a , P ) ) } ) ) ), ( @ ( fq ( 1 , a , P ) )
(  A ( { ( fq ( 1 , a , P ) ) } ) ) ), ( @ ( fq ( 1 , a , P ) ) (  A ( { ( fq ( 1 , a , P ) ) } ) ) ),
( @ ( fk ( 1 , a , P ) ) P ), ( @ ( fk ( 1 , a , P ) ) ( { ( fk ( 1 , a , P ) ) } ) ),
( @ ( fk ( 1 , a , P ) ) (  A ( { ( fk ( 1 , a , P ) ) } ) ) ), ( @ ( fk ( 1 , a , P ) )
(  A ( { ( fk ( 1 , a , P ) ) } ) ) ), ( @ ( fk ( 1 , a , P ) ) (  a ( { 1 } ) ) ),
( @ ( fk ( 1 , a , P ) ) (  a ( { ( fk ( 1 , a , P ) ) } ) ) ), ( @ ( fk ( 1 , a , P ) )
(  A ( { ( fk ( 1 , a , P ) ) } ) ) ), ( [ 1 = 1 ] ), ( [ ( fq ( 1 , a , P ) ) = ( fq ( 1 , a , P ) ) ] ),
( [ ( fk ( 1 , a , P ) ) = ( fk ( 1 , a , P ) ) ] ), ( ~ ( [ 1 = ( fq ( 1 , a , P ) ) ] ) ),
( @ 1 ( ~ ( { ( fq ( 1 , a , P ) ) } ) ) ), ( ~ ( [ ( fk ( 1 , a , P ) ) = 1 ] ) ),
( @ ( fk ( 1 , a , P ) ) ( ~ ( { 1 } ) ) ), ( ~ ( [ ( fk ( 1 , a , P ) ) = ( fq ( 1 , a , P ) ) ] ) ),
( @ ( fk ( 1 , a , P ) ) ( ~ ( { ( fq ( 1 , a , P ) ) } ) ) ), ( ~ ( [ 1 = ( fk ( 1 , a , P ) ) ] ) ),
( @ 1 ( ~ ( { ( fk ( 1 , a , P ) ) } ) ) ), ( ~ ( [ ( fq ( 1 , a , P ) ) = ( fk ( 1 , a , P ) ) ] ) ),
( @ ( fq ( 1 , a , P ) ) ( ~ ( { ( fk ( 1 , a , P ) ) } ) ) )]
```

```
//Give input for satisfiability check
@1 ~( ~(<q> a {12}) | ~( <k> a {12}) | (<x> a {12}) )

Unsatisfiable.
Contradiction: [( @ 1 ( ~ ( ~ (  a ( { 12 } ) ) ) ) ), ( @ 1 ( ~ ( ~ (  a ( { 12 } ) ) ) ) ),
( @ 1 ( ~ (  a ( { 12 } ) ) ) ), ( @ 1 ( ~ ( ( ~ (  a ( { 12 } ) ) ) | ( ~ (  a ( { 12 } ) ) ) ) ) ),
( @ 1 ( ~ ( ( ( ~ (  a ( { 12 } ) ) ) | ( ~ (  a ( { 12 } ) ) ) ) | (  a ( { 12 } ) ) ) ) ),
( @ 1 (  a ( { 12 } ) ) ), ( @ 1 (  a ( { 12 } ) ) ), ( @ 1 (  a ( { 12 } ) ) )]

@1 ~(~([!] a {12}) | {12})
Input file: Test120531191010/Input120531191915
Unsatisfiable.
Contradiction: [( @ 1 ( { 12 } ) ), ( @ 1 ( ~ ( { 12 } ) ) ), ( @ 1 ( ~ ( ~ ( [!] a ( { 12 } ) ) ) ) ), ( @ 1 ( ~ ( ( ~ (

@1 ~(~([!] a # P) | # P)
Input file: Test120531191010/Input120531192253
Unsatisfiable.
Contradiction: [( @ 1 ( # P ) ), ( @ 1 ( ~ ( # P ) ) ), ( @ 1 ( ~ ( ~ ( [!] a ( # P ) ) ) ) ), ( @ 1 ( ~ ( ( ~ ( [!] a (

@1 ~(~([? Q] a # P) | # P)
Input file: Test120531191010/Input120531192505
Unsatisfiable.
Contradiction: [( @ 1 ( # P ) ), ( @ 1 ( ~ ( # P ) ) ), ( @ 1 ( ~ ( ~ ( [? Q ] a ( # P ) ) ) ) ), ( @ 1 ( ~ ( ( ~ ( [? Q

@1 ~(~([? Q] a P) | P)
Input file: Test120531191010/Input120531192623
Satisfiable.
Model: [( @ 1 ( { 1 } ) ), ( @ 1 ( ~ P ) ), ( @ 1 ( ~ ( ~ ( [? Q ] a P ) ) ) ), ( @ 1 ( ~ ( ( ~ ( [? Q ] a P ) ) | P ) )

@1 ([?Q] a (<k> a P | <k> a ~P))
Input file: Test120531191010/Input120531193048
Satisfiable.
Model: [( @ 1 ( [? Q ] a (  a P ) ) ), ( @ 1 ( { 1 } ) ), ( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  A

@1 ([?Q] a ~(~(<x> a ~Q) | ~(<x> a Q)))
Input file: Test120531194002/Input120531194030
Satisfiable.
Model: [( @ 1 ( ~ ( ~ Q ) ) ), ( @ 1 ( ~ ( [? Q ] a Q ) ) ), ( @ 1 ( ~ ( ( ~ Q ) | ( [? Q ] a Q ) ) ) ),
( @ 1 (  a ( { 1 } ) ) ), ( @ 1 (  a ( { 1 } ) ) ), ( @ 1 (  a ( { 1 } ) ) ), ( @ 1 (  a ( ~ ( ( ~ Q ) |
( [? Q ] a Q ) ) ) ) ), ( @ 1 ( { 1 } ) ), ( @ 1 ( ~ ( [? Q ] a ( ~ (  a Q ) ) ) ) ),
( @ 1 ( ~ ( [? Q ] a ( ~ (  a ( ~ Q ) ) ) ) ) ), ( @ 1 ( ~ ( [? Q ] a ( ( ~ (  a ( ~ Q ) ) ) | ( ~ (  a Q ) ) ) ) ) ),
( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  A ( { 1 } ) ) ), ( @ 1 ( [? Q ] a ( ~ ( ( ~ (  a ( ~ Q ) ) )
( ~ (  a Q ) ) ) ) ) ), ( @ 1 ( [? Q ] a (  a Q ) ) ), ( @ 1 ( [? Q ] a (  a ( ~ Q ) ) ) ),
( @ 1 ( Q | (  a ( ~ ( Q | ( [? Q ] a Q ) ) ) ) ) ),
( @ 1 ( ( ~ Q ) | (  a ( ~ ( ( ~ Q ) | ( [? Q ] a Q ) ) ) ) ) ), ( [ 1 = 1 ] ), ( @ 1 Q )]

@1 (~(#P) | [?(#P)] A ~(<q> A ~(#P)))
Input file: Test120531194002/Input120531202324
Satisfiable.
Model: [( @ 1 ( ~ ( # P ) ) ), ( @ 1 ( { 1 } ) ), ( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  A ( { 1 }

@1 (~({12}) | [?{12}] A ~(<q> A ~({12})))
Input file: Test120531194002/Input120531202854
Satisfiable.
Model: [( ~ ( [ 12 = 1 ] ) ), ( @ 12 ( ~ ( { 1 } ) ) ), ( @ 1 ( { 1 } ) ), ( @ 1 (  A ( { 1 } ) ) ), ( @ 1 (  A ( { 1 } )

@1 ~(~(#P) | [?(#P)] A (~(<q> A ~(#P))))
Input file: Test120531194002/Input120531205512
Unsatisfiable.
Contradiction: [( @ 1 ( ~ ( ~ ( # P ) ) ) ), ( @ 1 ( ~ ( [? ( # P ) ] A ( ~ (  A ( ~ ( # P ) ) ) ) ) ) ), ( @ 1 ( ~ ( ( ~
```

A.5 Implementing a Tableau Prover for IEL in `Haskell`

Before diving into implementation details we continue the presentation of SQL with a concrete illustration questioning sequence having length two, the simplest structure. In this case the sequential reduction axiom gives rise to the following equivalence.

$$\begin{aligned}[\varphi_0,\varphi_1]R\varphi \quad \leftrightarrow \quad & \varphi_0 \wedge \ [\varphi_0]\varphi_1 \wedge R((\ \varphi_0 \wedge \ [\varphi_0]\varphi_1) \to [\varphi_0,\varphi_1]\varphi) \vee \\ & \varphi_0 \wedge \neg[\varphi_0]\varphi_1 \wedge R((\ \varphi_0 \wedge \neg[\varphi_0]\varphi_1) \to [\varphi_0,\varphi_1]\varphi) \vee \\ & \neg\varphi_0 \wedge \ [\varphi_0]\varphi_1 \wedge R((\neg\varphi_0 \wedge \ [\varphi_0]\varphi_1) \to [\varphi_0,\varphi_1]\varphi) \vee \\ & \neg\varphi_0 \wedge \neg[\varphi_0]\varphi_1 \wedge R((\neg\varphi_0 \wedge \neg[\varphi_0]\varphi_1) \to [\varphi_0,\varphi_1]\varphi)\end{aligned}$$

It is not difficult to see that in the reduction axiom above the disjunction pattern gives rise indeed to a partition, however, the questioning sequence does not.

Sometimes it is useful to allow more expressive power and use sequences of pairs of formulae and formula sequences. In such cases we use the notation $\sigma(n) = \langle \varphi_0/_{\sigma_0}, \ldots, \varphi_{n-1}/_{\sigma_{n-1}} \rangle$.

Of course there are many advantages in considering questioning sequences that correspond to (diachronic) partitions. We can also introduce a (in)dependence operator $\cdot/.$ to capture questioning sequences that correspond to (synchronic) partitions. For instance, given a sequence of length three that corresponds to a partition we have the following equivalence:

$$\begin{aligned}[\varphi_0,\varphi_1/_{\varphi_0},\varphi_2/_{\varphi_0,\varphi_1}]R\varphi \quad \leftrightarrow \quad & \varphi_0 \wedge R(\varphi_0 \to [\varphi_0,\varphi_1/_{\varphi_0},\varphi_2/_{\varphi_0,\varphi_1}]\varphi) \vee \\ & \varphi_1 \wedge R(\varphi_1 \to [\varphi_0,\varphi_1/_{\varphi_0},\varphi_2/_{\varphi_0,\varphi_1}]\varphi) \vee \\ & \varphi_2 \wedge R(\varphi_2 \to [\varphi_0,\varphi_1/_{\varphi_0},\varphi_2/_{\varphi_0,\varphi_1}]\varphi)\end{aligned}$$

The particular case of yes-no partition questions from DELQ can be obtained as a particular case of a synchronic partition sequence of length two:

$$[\varphi_0,\varphi_1/_{\varphi_0}]R\varphi \quad \leftrightarrow \quad \varphi_0 \wedge R(\varphi_0 \to [\varphi_0,\varphi_1/_{\varphi_0}]\varphi) \ \vee \ \varphi_1 \wedge R(\varphi_1 \to [\varphi_0,\varphi_1/_{\varphi_0}]\varphi)$$

by applying the following substitution $\varphi_1 \mapsto \neg\varphi_0, \varphi \mapsto \psi$:

$$[\varphi_0,\neg\varphi_0/_{\varphi_0}]R\varphi \quad \leftrightarrow \quad \varphi_0 \wedge R(\ \ \varphi_0 \to [\varphi_0,\neg\varphi_0/_{\varphi_0}]\psi) \ \vee \ \varphi_1 \wedge R(\neg\varphi_0 \to [\varphi_0,\neg\varphi_0/_{\varphi_0}]\varphi).$$

We proceed now towards discussing the theoretical background behind the functionality contained in each of the modules and simultaneously introducing the concrete implementation details. The level of detail in which implementation details are discussed, in general, mirrors their theoretical relevance. When discussing code constructs we also refer to their context and include the number of the code line where they are introduced.

A.6 The `Syntax.lhs` Module

The first module contains the main syntactic constructs used in specifying and synthesizing the tableau calculus. We start with the hybrid modal and dynamic specification object language, defined starting at line 14. This contains the nominals and propositional sorts as well as the static query, epistemic and interaction modalities and the dynamic questioning and resolution modalities.

```
1  module Syntax where
2
3  type Agent = Id
```

```
4  type Seq0 = [Form]
5  type Seq1 = [(Form,Seq0)]
6  type Seq2 = [(Seq1,(([Seq1],[Agent]),([Seq1],[Agent])))]
7
8  answerDep :: [a] -> Int -> a
9  answerDep s n = s!!n
10
11 questionIqqAcc :: [(a, b)] -> Int -> a
12 questionIqqAcc s n = fst (s!!n)
13
14 type Id = Integer
15 data Nom = C Id | N Id | V Id deriving (Eq,Ord)
16 data Prop = P Id | Q Id deriving (Eq,Ord)
17 data Form =  Bool Bool | Prop Prop | Nom Nom | Neg Form | Impl Form Form
18   | Conj [Form] | Disj [Form] | Box Rel Form | Dia Rel Form | Sat Nom Form
19   | Set Nom | Atom Prop | Quest Seq0 Form | Reso Form   deriving (Eq,Ord)
20
21 type Rel = Integer
22 data DomS = D Id | X Id | Y Id | Z Id deriving (Eq,Ord)
23 data Term = DomS DomS | Fun0 Nom | Fun1 Term | Fun2 Term | Fun3 Term
24           | Fun77 Term | Fun22 Term Term deriving (Eq,Ord)
25 data FOF = FBool Bool | SRel Id Term Term | Holds Form Term | Mark Id
26   | FNeg FOF | FConj [FOF] | FDisj [FOF] ———  | FImpl FOF FOF
27   | Equal Term Term | Forall DomS FOF | Exists DomS FOF  deriving (Eq,Ord)
28
29 type Node = [FOF]
30 type Tableau = [Node]
```

The second code block, starting at line 21, introduces the syntactic components in the first-order specification meta-language. This contains the domain sort, the "holds" predicate, an interpretation function for nominals, and functions and predicates that are used in lifting semantics to the modal logic connectives.

The final components in the module are the data structures for nodes, line 29, and tableaux, line 30. A node in a tableau is represented as a list of formulae and a tableau is a list of nodes. By construction, these lists have further structure, for instance, formulas inside a node are ordered by their syntactic structure, which is used later during expansion.

A.7 The `Qtab.lhs` Module

This module contains the main tableau functionality. We start, at line 2, by importing some basic list processing functionality to be used later on and by importing all the other modules, line 3. The `decide` function, line 5, provides the main functionality, it takes a formula and returns `True` if all branches of its corresponding tableau are closed, it also returns the number of steps needed to build the fully expanded tableau.

In order to retrieve further information about the structure of the tableau and the decomposition process the function `analyze` can be used, line 11. It returns a list containing, for all expansion steps their number, the "closed" and "expanded" decisions as Booleans and the content of the resulting tableau as a list of nodes.

The "closed" decision is made by the function at line 36 which takes a list of nodes as formula lists and checks it for contradictions using the auxiliary functionality provided by the functions at lines 21, 4, 14 and 17.

```
1  module Sql where
2  import Data.List (nub, zip4)
```

```
3  import Syntax; import Decomp; import Backgrd; import Divide; import Auxilar
4
5  decide :: Form -> (Bool)
6  decide f = trd (last (analyze f))
7
8  deciden :: Form -> (Bool, Integer)
9  deciden f = (trd (last (analyze f)), fst4 (last (analyze f)))
10
11 analyze :: Form -> [(Integer, Bool, Bool, Tableau)]
12 analyze f = take (1 + length (takeWhile (not . snd4) (zip4 [0 :: Integer ..]
13    (map (exhausted . (\ x -> expand (snd x) (fst x))) (zip [0 ..] (repeat f)))
14    (map (closed .    (\ x -> expand (snd x) (fst x))) (zip [0 ..] (repeat f)))
15    (map (\x -> expand (snd x) (fst x)) (zip [0..] (repeat f)))) )) (zip4 [0..]
16    (map (exhausted . (\ x -> expand (snd x) (fst x))) (zip [0 ..] (repeat f)))
17    (map (closed . (\ x -> expand (snd x) (fst x))) (zip [0 ..] (repeat f)))
18    (map (\x -> expand (snd x) (fst x)) (zip [0..] (repeat f))))
19
20 expandTableau :: Tableau -> Tableau
21 expandTableau = foldr ((++) . expandBranch) []
22
23 expandBranch :: Node -> Tableau
24 expandBranch n | all (\ x -> atom x || mark x) n = [n]
25                | line  (head (divide n)) = map ((nub . divide) .
26                        (\ x -> tail n ++ x)) (decompose (head n) (tail n))
27                | fresh (head (divide n)) = map ((nub . divide) . (\ x -> x ++ tail n))
28             (let n0 = tail n ++ head (decompose (head n) (tail n));
29                  n1 = nub (nomicol n0 ++ n0); n2 = nub (equality n1 ++ n1);
30                  n3 = nub (valuation n2 ++ n2); n4 = nub (epistemic n3 ++ n3);
31                  n5 = nub (intersection n4 ++ n4); n6 = nub (urBlockng n5 ++ n5)
32               in [nub (divide n6)])
33                | otherwise = map ((nub . divide). (\ x -> x ++ tail n ) )
34       (decompose (head n) (tail n))
35
36 closed :: Tableau -> Bool
37 closed = all contradiction
```

The "closed" decision is made by the function at line 36 which takes a list of nodes as formulae lists and checks it for contradictions using the auxiliary functionality provided by the functions at lines 21, 4, 14 and 17 from the `Auxilar` module in Section A.12.

Correspondingly, the function at line 49 decides whether or not a tableau is fully expanded by mapping the auxiliary function defined at line 46 over all the nodes in the tableau. Expansion for a given number of steps only is performed by the function from line 39. The tableau expansion, line 20, proceeds by expanding its component branches one by one, line 23.

```
39 expand :: Form -> Int -> Tableau
40 expand f 0 = let n0 = (head (initiate f)); n1 = nub (nomicol n0 ++ n0);
41                 n2 = nub (equality n1 ++ n1); n3 = nub (valuation n2 ++ n2);
42                 n4 = nub (epistemic n3 ++ n3); n5 = nub (intersection n4 ++ n4);
43                 n6 = nub (urBlockng n5 ++ n5) in [nub (divide n6)]
44 expand f n = expandTableau (expand f (n-1))
45
46 expanded :: Node -> Bool
47 expanded = all (\ x -> atom x || mark x)
48
49 exhausted :: Tableau -> Bool
50 exhausted = all expanded
```

A.8 The `Decomp.lhs` Module

The tableau expansion process defined in the previous module proceeds by logical decomposition rules. The current module defines the functionality needed for expanding tableau branches by means of logical decomposition of component formulae in their constituent subformulae.

```
1  module Decomp where
2  import CombinatoricsGeneration(cartProd); import Data.List(inits,(\\))
3  import Syntax; import Auxilar
4
5  ominus s1 s2 = s1 \\ s2
6  delta s = ominus (map fst (init s)) (snd (last s))
7  d2seq1 s = zip (delta s) (repeat [])
8
9  subseq :: [Form] -> [[[([Form], Form)]]]
10 subseq s = cartProd (map (\x-> cartProd [[([Bool False],Bool False),([Bool True],Bool True)],[x]]) (zip (inits s) s))
11
12 liter :: [([Form], Form)] -> Form
13 liter l | snd (l!!0)==Bool False = Quest (fst (l!!1)) (snd (l!!1))
14        |otherwise = Neg (Quest (fst (l!!1)) (snd (l!!1)))
15 --        | snd (l!!0)==Bool True = Neg (Quest (fst (l!!1)) (snd (l!!1)))
16
17 boolSubseq :: [Form] -> [[Form]]
18 boolSubseq l = map (map (\x-> liter x)) (subseq l)
19
20 conjSubseq :: [Form] -> [Form]
21 conjSubseq l = map Conj (boolSubseq l)
22
23 decompose :: FOF -> [FOF] -> [[FOF]]
24 decompose (Holds (Prop p) x) _ = [[(Holds (Prop p) x)]]
25 decompose (FNeg (Holds (Prop p) x)) _ = [[FNeg (Holds (Prop p) x)]]
26 decompose (Holds (Neg (Prop p)) x) _ = [[FNeg (Holds (Prop p) x)]]
27 decompose (Holds (Nom n) x) _ = [[Holds (Nom n) x]]
28 decompose (FNeg (Holds (Nom n) x)) _ = [[FNeg (Holds (Nom n) x)]]
29 decompose (Holds (Neg (Nom n)) x) _ = [[FNeg (Holds (Nom n) x)]]
30 decompose (Equal t1 t2) _ = [[Equal t1 t2]]
31 decompose (FNeg (Equal t1 t2)) _ = [[FNeg (Equal t1 t2)]]
32 decompose (FNeg (FNeg (Equal t1 t2))) _ = [[Equal t1 t2]]
33 decompose (SRel n t1 t2) _ = [[SRel n t1 t2]]
34 decompose (FNeg (SRel n t1 t2)) _ = [[FNeg (SRel n t1 t2)]]
35 decompose (FNeg (FNeg (SRel n t1 t2))) _ = [[SRel n t1 t2]]
36 decompose (Holds (Bool b) x) _ = [[Holds (Bool b) x]]
37 decompose (FNeg (Holds (Bool b) x)) _ = [[FNeg (Holds (Bool b) x)]]
38 decompose (Holds (Neg (Bool b)) x) _ = [[FNeg (Holds (Bool b) x)]]
```

The main function of the module is `decompose`, line 23, which specifies for every possible formula type its corresponding logical decomposition. In the first code block we deal with the atomic components. These can be either nominal atoms, line 7, propositional atoms, line 24, or atomic expressions of equality between terms, line 30, or binary applications of the relational predicates to terms, line 33. We also have boolean atoms, line 36.

```
39 decompose (Holds (Conj l) x) _ = [map (`Holds` x) l]
40 decompose (Holds (Neg (Conj l)) x) _ = map (\z -> [Holds (Neg z) x]) l
41 decompose (Holds (Disj l) x) _ = map (\z -> [Holds z x]) l
42 decompose (Holds (Neg (Disj l)) x) _ = [map (\ z -> Holds (Neg z) x) l]
43 decompose (Holds (Impl f1 f2) x) _ = [[Holds (Neg f1) x],[Holds f2 x]]
44 decompose (Holds (Neg (Impl f1 f2)) x) _ = [[Holds f1 x,Holds (Neg f2) x]]
45 decompose (Holds (Sat n f) _) _ = [[Holds f (Fun0 n)]]
46 decompose (Holds (Neg (Sat n f)) _) _ = [[Holds (Neg f) (Fun0 n)]]
```

```
47 decompose (Holds (Quest _ (Prop p)) x) _ = [[Holds (Prop p) x]]
48 decompose (Holds (Neg (Quest _ (Prop p))) x) _ = [[Holds (Neg (Prop p)) x]]
49 decompose (Holds (Reso (Prop p)) x) _ = [[Holds (Prop p) x]]
50 decompose (Holds (Neg (Reso (Prop p))) x) _ = [[Holds (Neg (Prop p)) x]]
51 decompose (Holds (Quest _ (Nom n)) x) _ = [[Holds (Nom n) x]]
52 decompose (Holds (Neg (Quest _ (Nom n))) x) _ = [[Holds (Neg (Nom n)) x]]
53 decompose (Holds (Reso (Nom n)) x) _ = [[Holds (Nom n) x]]
54 decompose (Holds (Neg (Reso (Nom n))) x) _ = [[Holds (Neg (Nom n)) x]]
55 decompose (Holds (Quest _ (Bool b)) x) _ = [[Holds (Bool b) x]]
56 decompose (Holds (Reso (Bool b)) x) _ = [[Holds (Bool b) x]]
57 decompose (Holds (Quest l (Neg f)) x) _ = [[Holds (Neg (Quest l f)) x]]
58 decompose (Holds (Reso (Neg f)) x) _ = [[Holds (Neg (Reso f)) x]]
```

The boolean connectives are dealt with as follows: conjunction at line 39, disjunction at line 41, and implication at line 43. The satisfaction operator is decomposed at line 45.

The decomposition rules for the questioning modalities depend on the logical structure of the formula inside the scope of the questioning modality. These can be applyed to propositional atoms, line 47, nominals, line 51, boolean atoms, line 55 or negated formulae 57.

```
59 decompose (Holds (Dia 1 f) x) _ = [[SRel 1 x (Fun1 x), Holds f (Fun1 x)]]
60 decompose (Holds (Dia 2 f) x) _ = [[SRel 2 x (Fun2 x), Holds f (Fun2 x)]]
61 decompose (Holds (Dia 3 f) x) _ = [[SRel 3 x (Fun3 x), Holds f (Fun3 x)]]
62 decompose (Holds (Dia r f) x) _ = [[SRel r x (Fun77 x), Holds f (Fun77 x)]]
63 decompose (Holds (Neg (Dia r f)) x) _ = [[Holds (Box r (Neg f)) x]]
64 --decompose (Holds (Neg (Dia r f)) x) _ = [[Holds (Box r (Neg f)) x]]
65 decompose (Holds (Box 1 f) x) fs  = [map
66     (\ z -> FDisj [FNeg (SRel 1 x z), Holds f z]) (image x 1 (rels2pairs fs))]
67 decompose (Holds (Box 2 f) x) fs  = [map
68     (\ z -> FDisj [FNeg (SRel 2 x z), Holds f z]) (image x 2 (rels2pairs fs))]
69 decompose (Holds (Box 3 f) x) fs  = [map
70     (\ z -> FDisj [FNeg (SRel 3 x z), Holds f z]) (image x 3 (rels2pairs fs))]
71 decompose (Holds (Box r f) x) fs  = [map
72     (\ z -> FDisj [FNeg (SRel r x z), Holds f z]) (image x r (rels2pairs fs))]
73 --decompose (Holds (Neg (Box r f)) x) _ = [[Holds (Dia r (Neg f)) x]]
74 decompose (Holds (Neg (Box 1 f)) x) _ = [[SRel 1 x (Fun1 x), Holds (Neg f) (Fun1 x)]]
75 decompose (Holds (Neg (Box 2 f)) x) _ = [[SRel 2 x (Fun2 x), Holds (Neg f) (Fun2 x)]]
76 decompose (Holds (Neg (Box 3 f)) x) _ = [[SRel 3 x (Fun3 x), Holds (Neg f) (Fun3 x)]]
77 decompose (Holds (Neg (Box r f)) x) _ = [[SRel r x (Fun77 x), Holds (Neg f) (Fun77 x)]]
```

Next come the modal formulae for each of the basic static questioning modalities: questions, line 59, knowledge 60 and intersection 61, as well as generic modalities 62. These use the semantic definitions of the modal connectives and introduce new terms. Correspondingly, the uiversal modalities are decomposed at lines 65, 67, 69 and 71 following their semantic definitions, by tableau rules branching into relational and propositional disjuncts.

The logical connectives in the first-order specification metalanguage are treated next. Conjunctions, line 151, and disjunctions, 153, are standard, the double negation needs to consider both oject- and metalanguage levels, line 155.

```
78 decompose (Holds (Quest [] f) x) _ = [[Holds f x]]
79 decompose (Holds (Neg (Quest [] f)) x) _ = [[Holds (Neg f) x]]
80 decompose (Holds (Quest _ (Prop p)) x) _ = [[Holds (Prop  p) x]]
81 decompose (Holds (Neg (Quest _ (Prop p))) x) _ = [[Holds (Neg (Prop  p)) x]]
82 decompose (Holds (Quest l (Neg f)) x) _ = [[Holds (Neg (Quest l f)) x]]
83 decompose (Holds (Neg (Quest l (Neg f))) x) _ = [[Holds ((Quest l f)) x]]
84 decompose (Holds (Quest l1 (Conj l)) x) _ = [[FConj (map
85     (\ y -> (Holds (Quest l1 y) x)) l)]]
86 decompose (Holds (Neg (Quest l1 (Conj l))) x) _ = map (\y -> [Holds (Neg (Quest l1 y)) x]) l
87 decompose (Holds (Reso (Conj l)) x) _ = [[FConj (map
```

```
88      (\ y -> Holds (Reso y) x) l)]]
89  decompose (Holds (Neg (Reso (Conj l))) x) _ = map (\y -> [Holds (Neg (Reso y)) x]) l
90  decompose (Holds (Quest l1 (Disj l)) x) _ = map (\y -> [Holds (Quest l1 y) x]) l
91  decompose (Holds (Neg (Quest l1 (Disj l))) x) _ = [[FConj (map
92      (\ y -> (Holds (Neg (Quest l1 y)) x)) l)]]
93  decompose (Holds (Reso (Disj l)) x) _ = map (\y -> [Holds (Reso y) x]) l
94  decompose (Holds (Neg (Reso (Disj l))) x) _ = [[FConj (map
95      (\ y -> Holds (Neg (Reso y)) x) l)]]
96  decompose (Holds (Quest f1 (Impl f2 f3)) x) fs = decompose (Holds        fs)
97    (Quest f1 (Disj [Neg f2, f3])) x) fs
98  decompose (Holds (Neg (Quest f1 (Impl f2 f3))) x) fs = decompose (Holds
99    (Quest f1 (Conj [f2, Neg f3])) x) fs
100 decompose (Holds (Reso (Impl f2 f3)) x) fs = decompose (Holds (Reso (
101   Disj [Neg f2, f3])) x) fs
102 decompose (Holds (Neg (Reso (Impl f2 f3))) x) fs = decompose (Holds (Reso (
103   Conj [f2, Neg f3])) x) fs
104
105 decompose (Holds (Quest f1 (Reso (Prop p))) x) fs = decompose (Holds
106   (Quest f1 (Prop p)) x) fs
107 decompose (Holds (Quest f1 (Reso (Nom n))) x) fs = decompose (Holds
108   (Quest f1 (Nom n)) x) fs
109 decompose (Holds (Neg (Quest f1 (Reso (Prop p)))) x) fs = decompose (Holds
110   (Neg (Quest f1 (Prop p))) x) fs
111 decompose (Holds (Neg (Quest f1 (Reso (Nom n)))) x) fs = decompose (Holds
112   (Neg (Quest f1 (Nom n))) x) fs
113 decompose (Holds (Quest f1 (Reso (Neg f))) x) fs = decompose (Holds
114   (Quest f1 (Neg (Reso f))) x) fs
115 decompose (Holds (Neg (Quest f1 (Reso (Neg f)))) x) fs = decompose (Holds
116   (Neg (Quest f1 (Neg (Reso f)))) x) fs
117 decompose (Holds (Quest f1 (Reso (Conj l))) x) fs = decompose (Holds
118   (Quest f1 (Conj (map (\y -> Reso y) l))) x) fs
119 decompose (Holds (Neg (Quest f1 (Reso (Conj l)))) x) fs = decompose (Holds
120   (Neg (Quest f1 (Conj (map (\y -> Reso y) l)))) x) fs
121
122 decompose (Holds (Quest f1 (Reso (Box 1 f))) x) fs = decompose (Holds
123   (Quest f1 (Box 1 (Reso f))) x) fs
124 decompose (Holds (Quest f1 (Reso (Box 3 f))) x) fs = decompose (Holds
125   (Quest f1 (Box 3 (Reso f))) x) fs
126 decompose (Holds (Quest f1 (Reso (Box 2 f))) x) fs = decompose (Holds
127   (Quest f1 (Box 3 (Reso f))) x) fs
128 decompose (Holds (Neg (Quest f1 (Reso (Box 1 f)))) x) fs = decompose (Holds
129   (Neg (Quest f1 (Box 1 (Reso f)))) x) fs
130 decompose (Holds (Neg (Quest f1 (Reso (Box 3 f)))) x) fs = decompose (Holds
131   (Neg (Quest f1 (Box 3 (Reso f)))) x) fs
132 decompose (Holds (Neg (Quest f1 (Reso (Box 2 f)))) x) fs = decompose (Holds
133   (Neg (Quest f1 (Box 3 (Reso f)))) x) fs
134
135 decompose (Holds (Quest l (Box 1 f)) x) _ = [[Holds
136    (Disj (map (\y -> (Conj [y, Box 1 (Impl y (Quest l f))])) (conjSubseq l))) x]]
137 decompose (Holds (Neg (Quest l (Box 1 f))) x) _ = [[Holds
138    (Conj (map (\y -> (Neg (Conj [y, Box 1 (Impl y (Quest l f))]))) (conjSubseq l))) x]]
139 decompose (Holds (Quest l (Box 3 f)) x) _ = [[Holds
140    (Disj (map (\y -> (Conj [y, Box 3 (Impl y (Quest l f))])) (conjSubseq l))) x]]
141 decompose (Holds (Neg (Quest l (Box 3 f))) x) _ = [[Holds
142    (Conj (map (\y -> (Neg (Conj [y, Box 3 (Impl y (Quest l f))]))) (conjSubseq l))) x]]
143 decompose (Holds (Quest l (Box 2 f)) x) _ = [[Holds (Box 2 (Quest l f)) x]]
144 decompose (Holds (Neg (Quest l (Box 2 f))) x) _ = [[Holds (Neg (Box 2 (Quest l f))) x]]
145 decompose (Holds (Reso (Box 2 f)) x) _ = [[Holds (Box 3 (Reso f)) x]]
146 decompose (Holds (Neg (Reso (Box 2 f))) x) _ = [[Holds (Neg (Box 3 (Reso f))) x]]
147 decompose (Holds (Reso (Box 1 f)) x) _ = [[Holds (Box 1 (Reso f)) x]]
148 decompose (Holds (Neg (Reso (Box 1 f))) x) _ = [[Holds (Neg (Box 1 (Reso f))) x]]
```

```
149 decompose (Holds (Reso (Box 3 f)) x) _ = [[Holds (Box 3 (Reso f)) x]]
150 decompose (Holds (Neg (Reso (Box 3 f))) x) _ = [[Holds (Neg (Box 3 (Reso f))) x]]
151 decompose (FConj l) _ = [l]
152 decompose (FNeg (FConj l)) _ = map (\x -> [FNeg x]) l
153 decompose (FDisj l) _ = map (: []) l
154 decompose (FNeg (FDisj l)) _ = [map FNeg l]
155 decompose (Holds (Neg (Neg f)) x) _ = [[Holds f x]]
156 decompose (FNeg (FNeg (Holds f x))) _ = [[Holds f x]]
157 decompose (FNeg (Holds (Neg f) x)) _ = [[Holds f x]]
158 decompose (FNeg (FNeg f)) _ = [[f]]
159 decompose (FNeg f) _ = [[FNeg f]]
160 decompose _ _ = [[Mark 77]]
```

The final and most relevant decomposition cases are the ones synthesized from the reduction axioms. As the effects of the dynamic actions are different on the underlying relations for the static modalities the decomposition rules have a different structure. The questioning action applied to the issue relation requires a disjunction, line 135. For the epistemic modality we have a commutating behaviour, line 143 and for the interaction modality a disjunctive pattern again, line 139. The dynamic modalities interact with the conjunction, line 91, disjunction, line 90, and implication, line 96, in the expected way.

A.9 The `Backgrd.lhs` Module

The current module contains the background theory related functions. They are applied when the tableau is initialized and whenever the third clause of the branch expanding function at line23 returns a matching pattern, this also means that a new term has been introduced.

The first two functions at lines 4 and 7 apply equality constraints to nominals, using auxiliary functionality defined by the functions at lines 28, 31, 34, 38, 41 in Section A.12.

```
1  module Backgrd where
2  import Data.List(nub); import Syntax; import Auxilar
3
4  nomicol :: [FOF] -> [FOF]
5  nomicol l = concatMap collapseq (nomiclasses l)
6
7  nominals :: [FOF] -> [FOF]
8  nominals = filter nominal
9
10 equality :: [FOF] -> [FOF]
11 equality l = nub (reflex (domain l) ++ symetry l ++ transit l)
12
13 valuation :: [FOF] -> [FOF]
14 valuation l = map (uncurry Holds) (valpar l)
15
16 valpar :: [FOF] -> [(Form, Term)]
17 valpar l = nub ([(x,y) | x <- propdom l, y <- domain l, z <- domain l,
18   Holds x z `elem` l, Equal z y `elem` l] ++
19   [(x,y) | x <- propdom l, y <- domain l, z <- domain l,
20   Holds x z `elem` l, Equal y z `elem` l] )
21
22 epistemic :: [FOF] -> [FOF]
23 epistemic l = nub (reflexive (domain l) ++ symetric l ++ transitive l)
```

Next the conditions ensuring that the equality between terms is an equivalence relation are applied, line 10, using the corresponding functions for symetry, line 44, transitivity, line 48, and reflexivity, line 52, defined in the `Auxilar` module in from Section A.12.

Since the previous step might have introduced new nominal equalities, the function at line 13 makes sure that indiscernable nominals have identical propositional valuations using the auxiliary functions from line 16 and lines 55 and 59 from the `Auxilar` module, Section A.12. The next function, line 22, also ensures that the background theory properties for the underlying relations are aplied, in this case we have a reflexive line 63, symetric, line 67, and thransitive relation, line 71 as defined in Section A.12.

A very important background theory component is the crucial property of intersection which is essential for the reduction axioms. The function at line 25 ensures that this is the case using the auxiliry functions from line 76 and line 82 in the `Auxilar` module, Section A.12.

```
25 intersection :: [FOF] -> [FOF]
26 intersection l = nub (map (uncurry (SRel 3)) (intersec l) ++
27   map (uncurry (SRel 1)) (intersback l) ++ map (uncurry (SRel 2)) (intersback l))
28
29 urBlockng :: [FOF] -> [FOF]
30 urBlockng l = recursivetermal (domain l)
31
32 recursivetermal :: [Term] -> [FOF]
33 recursivetermal [] = []
34 recursivetermal l = nub (termal l ++ recursivetermal (tail l))
```

A final component is the application of the unrestricted blocking mechanism, line 29, which is relevant for termination and uses the auxiliary function at line 32 as well as the extra functions defined in the `Auxilar` module at lines 88, 91 and 106, see Section A.12.

A.10 The `Divide.lhs` Module

The `Divide` module contains a modicum of functionality aimed at making the tableau construction more efficient. This is achieved by superimposing an order on the list of formulae based on their syntactic structures, line 4. The main classes of formulae are atomic ones, line 9, formulae that require the introduction of fresh terms, line 24, branching formulae, line 28, and non-branching ones, line 48. Finally, a set of marking formulae is maintained to facilitate the application of further heuristic principles.

```
1  module Divide where
2  import Syntax
3
4  divide :: [FOF] -> [FOF]
5  divide [] = []
6  divide l = filter line l ++ filter fresh l ++ filter split l ++
7             filter atom l ++ filter mark l
8
9  atom :: FOF -> Bool
10 atom (Holds (Prop _) _) = True
11 atom (FNeg (Holds (Prop _) _)) = True
12 atom (Holds (Nom _) _) = True
13 atom (FNeg (Holds (Nom _) _)) = True
14 atom (Equal _ _) = True
15 atom (FNeg (Equal _ _)) = True
16 atom (SRel{}) = True
17 atom (FNeg (SRel{})) = True
18 atom (Holds (Bool _) _) = True
19 atom (FNeg (Holds (Bool _) _)) = True
20 atom (FBool _) = True
21 atom (FNeg (FBool _)) = True
22 atom _ = False
23
```

```
fresh :: FOF -> Bool
fresh (Holds (Dia _ _) _) = True
fresh _ = False
```

Since the node expansion function is always applied to the head of the list the `divide` function ensures that nonbranching formulae are processes first. Then the fresh term rules are applyed, and each fresh term introduction also triggers an application of all the background theory rules plus the rule of unrestricted blocking.

```
split :: FOF -> Bool
split (Holds (Neg (Conj _)) _) = True
split (Holds (Disj _) _) = True
split (FNeg (FConj _)) = True
split (Holds (Impl _ _) _) = True
split (Holds (Box _ _) _) = True
split (FDisj _) = True
split (Holds (Quest _ (Disj _)) _) = True
split (Holds (Reso (Disj _)) _) = True
split _ = False

mark :: FOF -> Bool
mark (Mark _) = True
mark (FNeg (Mark _)) = True
mark (Forall _ _) = True
mark (FNeg (Forall _ _)) = True
mark (Exists _ _) = True
mark (FNeg (Exists _ _)) = True
mark _ = False

line :: FOF -> Bool
line f = ((atom f) || (fresh f) || (split f) || (mark f)) == False
```

Finally, the branching rules have the lowest priority: their application is always postponed until they are the only ones left. This is repeated until all the formulae in the list are atomic.

A.11 Illustrations of Using the `Qtab.lhs` Implementation

The first example illustrates a questioning sequence of length two combined with the static knowledge modality, which has a commutating behaviour:

```
 deciden (Quest [Prop (P 0), Prop (Q 0)] (Box 2 (Disj [Prop (P 0), Prop (Q 0)])))
(False,5)
it :: (Bool, Integer)
(0.08 secs, 10087116 bytes)
*Sql> dpl (analyze (Quest [Prop (P 0), Prop (Q 0)] (Box 2 (Disj [Prop (P 0), Prop (Q 0)]))))
(0,False,False,[[H({[p0,q0]?}[2](p0 v q0),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0]])
(1,False,False,[[H([2]{[p0,q0]?}(p0 v q0),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0]])
(2,False,False,[[(-R2(x0,x0) v H({[p0,q0]?}(p0 v q0),x0)),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0]])
(3,False,False,[[-R2(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0],[H({[p0,q0]?}(p0 v q0),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0)
(4,False,False,[[-R2(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0],[H({[p0,q0]?}p0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0]
(5,True,False,[[-R2(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0],[R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[R3(x0,x0)
it :: ()
(0.11 secs, 5284652 bytes)
```

The second example illustrates a questioning sequence of length two in combination with the static issue modality, which uses reduction axioms based on the disjunction pattern:

```
*Sql> deciden (Quest [Prop (P 0), Prop (Q 0)] (Box 1 (Disj [Prop (P 0), Prop (Q 0)])))
(False,14)
it :: (Bool, Integer)
(0.23 secs, 6327936 bytes)

*Sql> dpl (analyze (Quest [Prop (P 0), Prop (Q 0)] (Box 1 (Disj [Prop (P 0), Prop (Q 0)]))))
(0,False,False,[[H({[p0,q0]?}[1](p0 v q0),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0]])
(1,False,False,[[H(({{[]?}p0 & {[p0]?}q0) & [1](({[]?}p0 & {[p0]?}q0) -> {[p0,q0]?}(p0 v q0))) v (({[]?}p0 & -{[p0]?}q0)
(2,False,False,[[H(({{[]?}p0 & {[p0]?}q0) & [1](({[]?}p0 & {[p0]?}q0) -> {[p0,q0]?}(p0 v q0))),x0),R3(x0,x0),R1(x0,x0),R2
(3,False,False,[[H({{[]?}p0 & {[p0]?}q0),x0),H([1](({[]?}p0 & {[p0]?}q0) -> {[p0,q0]?}(p0 v q0)),x0),R3(x0,x0),R1(x0,x0),
(4,False,False,[[H({[]?}p0,x0),H({[p0]?}q0,x0),H([1](({[]?}p0 & {[p0]?}q0) -> {[p0,q0]?}(p0 v q0)),x0),R3(x0,x0),R1(x0,x0
(5,False,False,[[H({[p0]?}q0,x0),H([1](({[]?}p0 & {[p0]?}q0) -> {[p0,q0]?}(p0 v q0)),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0
(6,False,False,[[H([1](({[]?}p0 & {[p0]?}q0) -> {[p0,q0]?}(p0 v q0)),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),H(q
(7,False,False,[[(-R1(x0,x0) v H(({{[]?}p0 & {[p0]?}q0) -> {[p0,q0]?}(p0 v q0)),x0)),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,
(8,False,False,[[-R1(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),H(q0,x0)],[H(({{[]?}p0 & {[p0]?}q0) -> {[p0,q0]?
(9,False,False,[[-R1(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),H(q0,x0)],[H(-({{[]?}p0 & {[p0]?}q0),x0),R3(x0,x0
(10,False,False,[[-R1(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),H(q0,x0)],[H(-{[]?}p0,x0),R3(x0,x0),R1(x0,x0),R
,-H(q0,x0)]])
(11,False,False,[[-R1(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),H(q0,x0)],[H(-p0,x0),R3(x0,x0),R1(x0,x0),R2(x0,
x0),R2(x0,x0),x0=x0,H(q0,x0),-H(p0,x0)],[-R1(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,-H(p0,x0),-H(q0,x0)],[H(-(-{[]?}p
(12,False,False,[[-R1(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),H(q0,x0)],[R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,
x0,x0),x0=x0,-H(p0,x0),-H(q0,x0)],[H(--{[]?}p0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,-H(p0,x0),-H(q0,x0)],[H(--{[p0]?}q
(13,False,False,[[-R1(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),H(q0,x0)],[R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,
-H(p0,x0),-H(q0,x0)],[H({[]?}p0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,-H(p0,x0),-H(q0,x0)],[H({[p0]?}q0,x0),R3(x0,x0),R
(14,True,False,[[-R1(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),H(q0,x0)],[R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H
H(p0,x0),-H(q0,x0)],[R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,-H(p0,x0),-H(q0,x0),H(p0,x0)],[R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=
it :: ()
(0.53 secs, 15716944 bytes)
```

x0=x0, -

The next example illustrates a questioning sequence combining both asking actions and resolution actions. Because the resolution modality is idempotent, all resolution sequences are equivalent to a sequence of length one:

```
*Sql> deciden (Quest [Prop (P 0)] (Reso  (Box 2 (Prop (P 0)))))
(False,12)
it :: (Bool, Integer)
(0.16 secs, 5290132 bytes)

*Sql> dpl (analyze (Quest [Prop (P 0)] (Reso  (Box 2 (Prop (P 0))))))
(0,False,False,[[H({[p0]?}{!}[2]p0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0]])
(1,False,False,[[H(({{[]?}p0 & [3]({[]?}p0 -> {[p0]?}{!}p0)) v (-{[]?}p0 & [3](-{[]?}p0 -> {[p0]?}{!}p0))),x0),R3(x0,x0),
(2,False,False,[[H(({[]?}p0 & [3]({[]?}p0 -> {[p0]?}{!}p0)),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0],[H((-{[]?}p0 & [3](-
(3,False,False,[[H({[]?}p0,x0),H([3]({[]?}p0 -> {[p0]?}{!}p0),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0],[H(-{[]?}p0,x0),H(
(4,False,False,[[H({[]?}p0,x0),H([3]({[]?}p0 -> {[p0]?}{!}p0),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0],[H(-{[]?}p0,x0),H(
(5,False,False,[[H([3]({[]?}p0 -> {[p0]?}{!}p0),x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[H(-p0,x0),H([3](-{[]?}
(6,False,False,[[(-R3(x0,x0) v H(({[]?}p0 -> {[p0]?}{!}p0),x0)),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[H([3](-{[]
(7,False,False,[[-R3(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[H(({[]?}p0 -> {[p0]?}{!}p0),x0),R3(x0,x0),R1(x
(8,False,False,[[-R3(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[H(-{[]?}p0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x
(9,False,False,[[-R3(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[H(-{[]?}p0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x
(10,False,False,[[-R3(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[H(-p0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0
(11,False,False,[[-R3(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),
(12,True,False,[[-R3(x0,x0),R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0)],[R3(x0,x0),R1(x0,x0),R2(x0,x0),x0=x0,H(p0,x0),-
it :: ()
(0.28 secs, 8403804 bytes)
```

The final illustration shows the difference between knowing that and knowing whether. After a yes/no question about p the issue relation decides whether p, this turns out to be a SQL validity, however, it does not settle that p holds.

```
*Sql> (deciden (Neg (Quest [Prop (P 0),Neg (Prop (P 0))] (Box 1 (Prop (P 0))))))
(False,40)
it :: (Bool, Integer)
(0.84 secs, 63160048 bytes)

*Sql> (deciden (Neg (Quest [Prop (P 0),Neg (Prop (P 0))] (Disj [Box 1 (Prop (P 0)), Box 1 (Neg (Prop (P 0)))]))))
(True,84)
it :: (Bool, Integer)
(93.55 secs, 6657552984 bytes)
```

A.12 The `Auxilar.lhs` Module

The remaining code blocks contain functionality that has only an auxiliary role in the implementation. They are included here in order to make the file self contained as a literate programming `Haskell` script. Some of the functions have already been discussed in the main text.

```
module Auxilar where
import Data.List (intersect,nub,partition); import Syntax

content :: FOF -> FOF
content (FNeg (Equal t1 t2)) = Equal t1 t2
content (FNeg (SRel n t1 t2)) = SRel n t1 t2
content (FNeg (Holds f t)) = Holds f t
content (Equal t1 t2) = Equal t1 t2
content (SRel n t1 t2) = SRel n t1 t2
content (Holds f t) = Holds f t
content (FNeg f) = f
content (f) = f

negative :: [FOF] -> [FOF]
negative = filter negation

negation :: FOF -> Bool
negation (FNeg _) = True
negation _ = False

contradiction :: Node -> Bool
contradiction n = intersect (filter (not . negation) n) (
  map content (filter negation n)) /= []

nomiterms :: [FOF] -> Nom -> [Term]
nomiterms l n = [x | x <- domain l, Holds (Nom n) x `elem` l]

nomiclasses :: [FOF] -> [[Term]]
nomiclasses l = map (nomiterms l) (nomivals l)

nomivals :: [FOF] -> [Nom]
nomivals = concatMap nomi

nomi :: FOF -> [Nom]
nomi (Holds (Nom x) _) = [x]
nomi _ = []

collapseq :: [Term] -> [FOF]
collapseq l = map (uncurry Equal) [(x,y) | x <- l, y <- l, x == head l]
```

```
40
41  nomiclass :: [FOF] -> Nom -> [[FOF]]
42  nomiclass l n = [fst (partition (\x -> nomi x==[n]) l)]
43
44  symetry :: [FOF] -> [FOF]
45  symetry l = nub (map (uncurry Equal) [(x,y) | x <- domain l, y <- domain l,
46    Equal y x `elem` l])
47
48  transit :: [FOF] -> [FOF]
49  transit l = nub (map (uncurry Equal) [(x,z) | x <- domain l, z <- domain l,
50    y <- domain l, Equal x y `elem` l, Equal y z `elem` l])
51
52  reflex :: [Term] -> [FOF]
53  reflex = map (\ x -> Equal x x)
54
55  props :: FOF -> [Form]
56  props (Holds p _) = [p]
57  props _ = []
58
59  propdom :: [FOF] -> [Form]
60  propdom l = nub (concatMap props l)
61
62
63  reflexive :: [Term] -> [FOF]
64  reflexive l = map (\ x -> SRel 1 x x) l ++
65    map (\ x -> SRel 2 x x) l ++ map (\ x -> SRel 3 x x) l
66
67  symetric :: [FOF] -> [FOF]
68  symetric l = nub (map (\x -> SRel (fst3 x) (snd3 x) (trd3 x)) [(r,y,x) |
69    x <- domain l, y <- domain l, r <- map fst3 (rels2pairs l), SRel r x y `elem` l])
70
71  transitive :: [FOF] -> [FOF]
72  transitive l = nub (map (\x -> SRel (fst3 x) (snd3 x) (trd3 x)) [(r1,y,z) |
73     x <- domain l, y <- domain l, z <- domain l, r2 <- map fst3 (rels2pairs l),
74     r1 <- map fst3 (rels2pairs l), SRel r1 x y `elem` l, SRel r2 y z `elem` l, r1==r2])
75
76  intersec :: [FOF] -> [(Term,Term)]
77  intersec l = nub ([(x,y) | x <- domain l, y <- domain l, z <- domain l,
78     SRel 1 x y `elem` l, SRel 2 x z `elem` l, Equal y z `elem` l] ++
79     [(x,y) | x <- domain l, y <- domain l, z <- domain l,
80     SRel 1 x y `elem` l, SRel 2 x z `elem` l, Equal z y `elem` l])
81
82  intersback :: [FOF] -> [(Term,Term)]
83  intersback l = nub ([(x,y) | x <- domain l, y <- domain l, z <- domain l,
84     SRel 3 x z `elem` l, Equal y z `elem` l] ++
85     [(x,y) | x <- domain l, y <- domain l, z <- domain l,
86     SRel 3 x z `elem` l, Equal z y `elem` l])
87
88  termal :: [Term] -> [FOF]
89  termal l = map (\x -> FDisj [Equal (head l) x, FNeg (Equal (head l) x)]) (tail l)
90
91  terms :: FOF -> [Term]
92  terms (Holds _ t) = [t]
93  terms (FNeg (Holds _ t)) = [t]
94  terms (SRel _ t1 t2) = [t1,t2]
95  terms (FNeg (SRel _ t1 t2)) = [t1,t2]
96  terms (Equal t1 t2) = [t1,t2]
97  terms (FNeg (Equal t1 t2)) = [t1,t2]
98  terms (Mark _) = []
99  terms (FBool _) = []
100 terms (FNeg _) = []
```

```
terms (FConj _) = []
terms (FDisj _) = []
terms (Forall _ _) = []
terms (Exists _ _) = []

domain :: [FOF] -> [Term]
domain l = nub (concatMap terms l)

initiate :: Form -> Tableau
initiate (Disj l) = [[Holds (Disj l) (DomS (X 0))]]
initiate (Impl f1 f2) = [[Holds (Disj [Neg f1, f2]) (DomS (X 0))]]
initiate (Conj l) = [[Holds (Conj l) (DomS (X 0))]]
initiate (Neg f) = [[Holds (Neg f) (DomS (X 0))]]
initiate (Box r f) = [[Holds (Box r f) (DomS (X 0))]]
initiate (Dia r f) = [[Holds (Dia r f) (DomS (X 0))]]
initiate (Sat n f) = [[Holds ( Sat n f) (DomS (X 0))]]
initiate (Bool b) = [[Holds (Bool b) (DomS (X 0))]]
initiate (Prop p) = [[Holds (Prop p) (DomS (X 0))]]
initiate (Nom n) = [[Holds (Nom n) (DomS (X 0))]]
initiate (Quest f1 f2) = [[Holds (Quest f1 f2) (DomS (X 0))]]
initiate (Reso f) = [[Holds (Reso f) (DomS (X 0))]]

type Pairs = [(Id, Term, Term)]

rels2pairs :: [FOF] -> Pairs
rels2pairs = foldr ((++) . rel2pair) []

rel2pair :: FOF -> Pairs
rel2pair (SRel n t1 t2) = [(n,t1,t2)]
rel2pair _ = []

image :: Term -> Id -> Pairs -> [Term]
image _ _ [] = []
image x n r = nub (map snd (filter (\z -> fst z == x)
               [(v,w) | (z,v,w) <- r, z == n]))

nominal :: FOF -> Bool
nominal (Holds (Nom _) _) = True
nominal _ = False

showT :: Tableau -> [String]
showT = map show

display :: Tableau -> IO ()
display = putStrLn . init . unlines . showT

dpl :: Show a => [a] -> IO ()
dpl x = putStrLn (init (unlines (map show x)))

instance Show DomS where
   show (D n) = 'd': show n
   show (X n) = 'x': show n
   show (Y n) = 'y': show n
   show (Z n) = 'z': show n

instance Show Term where
   show (DomS d) = show d
   show (Fun0 n) = "F0" ++ "(" ++ show n ++ ")"
   show (Fun1 t) = "F1" ++ "(" ++ show t ++ ")"
   show (Fun2 t) = "F2" ++ "(" ++ show t ++ ")"
   show (Fun3 t) = "F3" ++ "(" ++ show t ++ ")"
```

```
162     show (Fun77 t) = "F77" ++ "(" ++ show t ++ ")"
163     show (Fun22 t1 t2) = "FF" ++ "(" ++ show t1 ++ "," ++ show t2 ++ ")"
164
165  instance Show FOF where
166     show (SRel r t1 t2) = "R" ++ show r ++ "(" ++ show t1 ++","++ show t2 ++ ")"
167     show (Holds f t) = "H"++"("++ show f ++","++ show t ++")"
168     show (Equal t1 t2) = show t1 ++"="++ show t2
169     show (FBool b) = show b
170     show (FNeg f)            = '-' : show f
171     show (FConj [])          = "T"
172     show (FConj [f])         = show f
173     show (FConj (f:fs)) = "("++ show f ++" & "++ showCtail fs ++ ")"
174       where showCtail []     = ""
175             showCtail [fm]    = show fm
176             showCtail (fm:fms) = show fm ++ " & " ++ showCtail fms
177     show (FDisj [])           = "F"
178     show (FDisj [f])          = show f
179     show (FDisj (f:fs))       =  "(" ++ show f ++ " v " ++ showDtail fs ++ ")"
180       where showDtail []     = ""
181             showDtail [fm]    = show fm
182             showDtail (fm:fms) = show fm ++ " v " ++ showDtail fms
183     show (Forall v f) = "A" ++ show v ++"("++ show f ++")"
184     show (Exists v f) = "E" ++ show v ++"("++ show f ++")"
185     show (Mark n)            = "Mk" ++ show n
186
187  instance Show Nom where
188     show (C n) = 'c': show n
189     show (V n) = 'x': show n
190     show (N n) = 'n': show n
191
192  instance Show Prop where
193     show (P n) = 'p': show n
194     show (Q n) = 'q': show n
195
196  instance Show Form where
197     show (Bool True) = "T"
198     show (Bool False) = "F"
199     show (Prop i)            = show i
200     show (Nom i)             = show i
201     show (Neg f)             = '-' : show f
202     show (Conj [])           = "T"
203     show (Conj [f])          = show f
204     show (Conj (f:fs)) = "(" ++ show f ++ " & " ++ showCtail fs ++ ")"
205       where showCtail []     = ""
206             showCtail [fm]    = show fm
207             showCtail (fm:fms) = show fm ++ " & " ++ showCtail fms
208     show (Disj [])           = "F"
209     show (Disj [f])          = show f
210     show (Disj (f:fs)) =  "(" ++ show f ++ " v " ++ showDtail fs ++ ")"
211       where showDtail []     = ""
212             showDtail [fm]    = show fm
213             showDtail (fm:fms) = show fm ++ " v " ++ showDtail fms
214     show (Impl f1 f2) = "(" ++ show f1 ++ " -> " ++ show f2 ++ ")"
215     show (Box name f)      = "[" ++ show name ++ "]" ++ show f
216     show (Dia name f)      = "<" ++ show name ++ ">" ++ show f
217     show (Sat nom f)         = '@': show nom ++ show f
218     show (Quest f1 f2) = "{" ++ show f1 ++ "?}" ++ show f2
219     show (Reso f) = "{!}" ++ show f
220
221  fst3 :: (t1,t2,t3) -> t1
222  fst3 (z,_,_) = z
```

```
223
224 snd3 :: (t1,t2,t3) -> t2
225 snd3 (_,z,_) = z
226
227 trd3 :: (t1,t2,t3) -> t3
228 trd3 (_,_,z) = z
229
230 fst4 :: (t1,t2,t3,t4) -> t1
231 fst4 (z,_,_,_) = z
232
233 snd4 :: (t1,t2,t3,t4) -> t2
234 snd4 (_,z,_,_) = z
235
236 trd :: (t1,t2,t3,t4) -> t3
237 trd (_,_,z,_) = z
238
239 lst :: (t1,t2,t3,t4) -> t4
240 lst (_,_,_,z) = z
241
242 translate :: Form  -> FOF
243 translate (Disj l) = Holds (Disj l) (DomS (X 0))
244 translate (Impl f1 f2) = Holds (Disj [Neg f1, f2]) (DomS (X 0))
245 translate (Conj l) = Holds (Conj l) (DomS (X 0))
246 translate (Neg f) = Holds (Neg f) (DomS (X 0))
247 translate (Box r f) = Holds (Box r f) (DomS (X 0))
248 translate (Dia r f) = Holds (Dia r f) (DomS (X 0))
249 translate (Sat n f) = Holds (Sat n f) (DomS (X 0))
250 translate (Bool b) = Holds (Bool b) (DomS (X 0))
251 translate (Prop p) = Holds (Prop p) (DomS (X 0))
252 translate (Nom n) = Holds (Nom n) (DomS (X 0))
253 translate (Quest f1 f2) = Holds (Quest f1 f2) (DomS (X 0))
254 translate (Reso f) = Holds (Reso f) (DomS (X 0))
```