Dynamic logic of questions
Minica, S.A.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

Download date: 16 Dec 2018
Chapter 5

Implementing Questioning Games

In this chapter we present and document the implementation behind the question answer games discussed in previous chapter. Some outstanding features of the implementation are the following: a representation strategies in game as questions in an epistemic model, computing resulting strategy profiles and intuitive display of formulae representing game-moves, computation of both local and global epistemic actions induced by questioning strategies and intuitive display for updated epistemic models, model checking of goal formulae in resulting epistemic structures, implementation of epistemic games with questioning moves and computation and intuitive display of the corresponding matrix for the induced game. The chapter presents and explains the literate Haskell (cf. [55, 68]) code of the QAGames.lhs module covering the theoretical aspects surrounding the epistemic games with questioning moves discussed in Chapter 4. Previous epistemic functionality from DEMo [107], DEMo-light [109], and DELq functionality from [71] is also explained and used in the current implementation. Finally, the PAGs.lhs module is used to compare questioning games with games with announcements.

In the last section we will define the birelataional coarsest partition problem and we will give an algorithm for minimizing issue-epistemic models using a notion of behavioral equivalence that is adequate for the questioning language.

5.1 Implementing Questioning Games

Much of the theoretical notions introduced so far and even more so much of the functionality contained in the discussed implementation have their utility enriched in applications involving various scenarios of strategic interaction between rational agents via dynamic questioning and informative actions.

This section implements extensions of basic epistemic functionality to model epistemic games with informative and questioning moves in the style of [2] and [1]. We will first model games in which moves of players are epistemic actions
which are public announcements triggered by mutual strategic questions, and in which the players seek to achieve goals represented by epistemic formulae.

We will use as the point of departure the standard dynamic epistemic functionality for dynamic epistemic logic from \textit{DEMo-light} \cite{109}. An introductory course presentation of both DEL and its basic Haskell implementation, in a version without vocabulary change, is also available in \cite{104}. The \textit{DEMo-light} modules in \cite{109} provide all the needed functionality to achieve the purpose of this section.

Staring from here we add an extension module that uses the PAL and DEL functionality for informative actions and builds the additional functionality needed to capture the strategic aspects involved in question answer games.

The working of the code is afterwards illustrated in all minuitia in Section 5.2 using a paradigmatic example. We will only present hereafter the main features of the code and use it for examples already discussed in Chapter 4.

The main utility provided by the code consists in taking an epistemic-goal structure with two main components a model and a pair of goal formulae and building from these the game matrix for the strategic question-answer game played in the given epistemic model with the given goals.

The following modules are imported by the main \texttt{QuestionGames.lhs} module:

\textbf{ModelsVocab} The module defines the basic data structures for epistemic models and epistemic formulae using an underlying propositional vocabulary and the underlying strong Kleene calculus.

\textbf{ActionVocab} The module defines the data structures for action models and uses it to implement action model update. The module also contains the implementation of public announcements.

\textbf{ChangeVocab} The module adds functionality for factual change and model checking for epistemic logic alongside with concrete illustrations for how to implement perception and information dynamics.

\textbf{ShortctsGms} The module contains various syntactic sugars useful for defining and working with question-specific notions, like, for instance, the tripartite extension used as the basis for strategy equivalence, the domain naming functionality, etc. It also contains additional useful functions that have only an ancillary role in display and related computations.

\subsection{The QAGames.lhs Question-Answer Games Module}

The module starts by importing background functionality as described before, lines 2-10. The next seven functions introduce utilities needed to define execution values for players’ strategies based on the structure of the epistemic model. First a formula is linked with a subset of states in the epistemic domain, line 12. The
second level also takes into account the indistinguishability relation for an agent when computing the relevant extension of a formula line 15. Given any model as input one can determine using nominals the number of formulae with different extensions, line 21, this also gives an upper bound to the number of strategies in the game. A questioning action lifts the extension of a formula to a set of three extensions corresponding to the triple of epistemically relevant answers, line 25.

```haskell
module QA Games where
import Control.Monad
import List
import Data.Ord (comparing)
import CombinatoricsGeneration
import qualified Data.Set as Set
import Shortc tsGms
import ModelsVocab hiding (m0)
import ActionVocab hiding (upd,public,preconditions,voc)
import ChangeVocab

extension :: (Ord a) => EpistM a -> Form -> Maybe [a]
extension m f = filterM (\x -> (isTrueAtMayb m x f)) (dom m)

knowsExtension3 :: (Ord state) => EpistM state -> Agent -> [[Maybe [state]]]
knowsExtension3 m ag =
  [[extension m (K ag x), extension m (Neg (Disj[(K ag x),
    (K ag (Neg x))])), extension m (K ag (Neg x))] | x <- (forms m)]

forms :: (Ord state) => EpistM state -> [Form]
forms m = map (\y -> Disj y) (map (\x -> noml m x)
  (powerList (dom m)))

formsK3 :: (Ord state) => EpistM state -> Agent -> [(Form, [Maybe [state]])]
formsK3 m ag = zip (forms m) (knowsExtension3 m ag)

formsK3Sort :: (Ord state) => EpistM state -> Agent -> [(Form, [Maybe [state]])]
formsK3Sort m ag = zip (forms m) (map sort
  (knowsExtension3 m ag))

formsK3Nuby :: (Ord state) => EpistM state -> Agent -> [(Form, [Maybe [state]])]
formsK3Nuby m ag = nubBy (\x y -> (snd y) ==
  (snd x)) (zip (forms m) (map sort (knowsExtension3 m ag)))
```

After sorting the tripartite extension of the epistemic answers, line 29, the upper bound of questioning strategies having non equivalent execution value in a model can be sensibly lowered by merging all formulae with equivalent execution value into only one representative strategy, line 34.
The following four functions use the notions implemented so far to compute global strategy profiles in the game. The first aspect concerns the fact that strategies are uniform inside the same information cell but might differ between elements of the partition, line 39. Next the individual strategies are globally aggregated to form strategy profiles for all the agents, line 45.

\[
\text{straGlob} :: (\text{Eq state}, \text{Ord state}, \text{Num state}) \Rightarrow \\
\text{EpistM state} \rightarrow \text{Agent} \rightarrow [ [\text{[state],Form}]] \\
\text{straGlob m ag} = \text{cartProd (map (\langle x \rightarrow \
\text{(zip (take (length (formsK3nuby m (aminus m ag))) (repeat x))) \
\text{(map fst (formsK3nuby m (aminus m ag)))) (infopart m ag))}) \\
\text{profGlob} :: (\text{Eq state}, \text{Ord state}, \text{Num state}) \Rightarrow \\
\text{EpistM state} \rightarrow [[[\text{[state],Form]]]] \\
\text{profGlob m} = \text{cartProd [straGlob m ((agents m) !! 0),} \\
\text{straGlob m (aminus m ((agents m) !! 0))]} \\
\text{w2infoCells} :: (\text{Eq state}, \text{Ord state}, \text{Num state}) \Rightarrow \\
\text{EpistM state} \rightarrow \text{state} \rightarrow [[\text{state}]] \\
\text{w2infoCells m w} = [\text{infocell m ((agents m)!!0) w}, \\
\text{infocell m ((agents m)!!1) w}] \\
\text{w_prof2forms} :: (\text{Eq state}, \text{Ord state}, \text{Num state}) \Rightarrow \\
\text{EpistM state} \rightarrow \text{state} \rightarrow [[[\text{[state],Form]]]} \rightarrow [\text{Form}] \\
\text{w_prof2forms m w p} = \text{map snd (filter (\langle x \rightarrow (\text{fst x}) =} \\
\text{infocell m ((agents m)!!0) w) (p!!0) \}} \\
\text{filter (\langle x \rightarrow (\text{fst x}) = infocell m ((agents m)!!1) w) (p!!1))} \\
\text{w_prof2exeVal} :: (\text{Eq state}, \text{Ord state}, \text{Num state}) \Rightarrow \\
\text{EpistM state} \rightarrow \text{state} \rightarrow [[[\text{[state],Form]]]} \rightarrow [\text{Form}] \\
\text{w_prof2exeVal m w p} = [\text{barval3 m w ((agents m)!!0)}, \\
\text{barval3 m w ((agents m)!!1)} \\
\text{((w_prof2forms m w p)!!0)}, \text{barval3 m w ((agents m)!!1)} \\
\text{((w_prof2forms m w p)!!1)}] \\
\text{w_prof2answ} :: (\text{Eq state}, \text{Ord state}, \text{Num state}) \Rightarrow \\
\text{EpistM state} \rightarrow \text{state} \rightarrow [[[\text{[state],Form]]]} \rightarrow \text{Form} \\
\text{w_prof2answ m w p} = \text{ Conj (w_prof2exeVal m w p)} \\
\text{w_prof2upd} :: (\text{Eq state}, \text{Ord state}, \text{Num state}) \Rightarrow \\
\text{EpistM state} \rightarrow \text{state} \rightarrow [[[\text{[state],Form]]]} \rightarrow \text{EpistM state} \\
\text{w_prof2upd m w p} = \text{ upd_pa m (w_prof2answ m w p)} \\
\text{w2updates} :: (\text{Eq state}, \text{Ord state}, \text{Num state}) \Rightarrow \\
\text{EpistM state} \rightarrow \text{state} \rightarrow [\text{EpistM state}] \\
\text{w2updates m s} = \text{map (\langle x \rightarrow (w_prof2upd m s x)) (profGlob m)}
The result is a conjunction of execution values, one for each agent, line 67. And each answer computed as a conjunction of execution values leads to a corresponding update of the initial model, as computed starting from line 71.

Taking each state of the model as the point of departure, each global strategy profile can be executed in it, leading to a list of corresponding updates, line 75.

```haskell
w2pays :: (Eq state, Ord state, Num state) =>
        EpistM state -> state -> Form -> [Integer]
w2pays m w g = map (\x -> paynumber x w g) (w2updates m w)

prof2w_pays :: (Eq state, Ord state, Num state) =>
                EpistM state -> [[[state],Form]] -> Form -> [Integer]
prof2w_pays m p g =
        map (\x -> paynumber (w_prof2upd m x p) x g) (dom m)

prof2w_paysum :: (Eq state, Ord state, Num state) =>
                   EpistM state -> [[[state],Form]] -> Form -> Integer
prof2w_paysum m p g = foldr (+) 0 (prof2w_pays m p g)
```

The next and final stage in constructing the game matrix consists in assigning payoffs to game outcomes, this is done by the following four functions. The payoff value is determined at each world by model checking agents’ goal formulae in the resulting updated model, line 79. The result of model checking goal formulae can be lifted from local states to a list of worlds in the domain and global strategy profiles, line 83. The payoff values of local strategy execution are aggregated in a global sum corresponding to each strategy profile, computed at line 88.

Finally, the `qagn` function computes the resulting entries in the matrix induced by the question-answer game played in the given epistemic model, line 92.

```haskell
qagn :: (Eq state, Ord state, Num state) =>
       EpistM state -> (Form,Form) -> [(Integer,Integer)]
qagn m g = zip (map (\x -> prof2w_paysum m x (fst g)) (profGlob m))
            (map (\x -> prof2w_paysum m x (snd g)) (profGlob m))
```

Two additional auxiliary functions, which both assume a named epistemic model as input, are used to compute the epistemic projection of a formula, first as a bipartite epistemic announcement, line 98, and second as a tripartite epistemic answer to a question, starting from line 101. This pair of function also give a concise comparison point between games played with spontaneous announcement moves and games played with questioning moves followed by answers.

```haskell
barval2 :: (Ord state) => EpistM state -> state -> Agent -> Form -> Form
barval2 m s ag f | isTrueAtMayb m s (K ag f) == Just True = K ag f
                  | otherwise = Neg (K ag f)

barval3 :: (Eq state, Ord state) =>
            EpistM state -> state -> Agent -> Form -> Form
barval3 m s ag f | isTrueAtMayb m s (K (aminus m ag) f) == Just True =
                   K (aminus m ag) f
```
Chapter 5. Implementing Questioning Games

5.1.2 The PAGs.lhs Extension Module

The infrastructure used to implement games with questioning actions uses answers as informative actions in their simplest form as public announcements. In this section we will show how this infrastructure can be extended to accommodate games with informative epistemic games. We present the remaining ancillary functionality as a separate module merely for ease of exposition, it is in fact a constitutive part of the QAGames.lhs module, sharing much of the functionality.

The `infopart` function reconstructs the information partition of a given agent in an epistemic model `m`, line 111. Next, using the function at line 114, the information cell determined by an additional state parameter can be retrieved.

At line 119 and following the information cell is mapped over the list of states in the domain, this will be useful for further processing in the next steps. The first processing step converts the information cells from sets of states to lists of nominals for the states, line 125. This list is further processed by mapping a disjunction over the nominals in the function starting from line 129.
Finally, the bipartite epistemic value of the disjunctions is computed by mapping model checking operations over the previous list, line 133.

```
realcellstratsK :: (Eq state, Ord state, Num state) =>
  EpistM state -> Agent -> [state] -> [Form]
realcellstratsK m ag c = cellpartdisKbar m ag (c!!0)

strategiesKK :: (Eq state, Ord state, Num state) =>
  EpistM state -> Agent -> [[(state,Form)]]
strategiesKK m a = cartProd (map (\x ->
  (zip (take (length (realcellstratsK m a x)) (repeat x))
  (realcellstratsK m a x))) (infopart m a))

profilesKK :: (Eq state, Ord state, Num state) =>
  EpistM state -> [[[(state,Form)]]]
profilesKK m = cartProd [strategiesKK m ((agents m) !! 0),
  strategiesKK m (aminus m ((agents m) !! 0))]
```

Taking only one representative state instead of an entire partition cell simplifies further computations by preserving only the minimally required information, line 136. Global strategies are computed at line 140 for a model and a player given as inputs by assigning to every cell in the information partition a formula.

This ensures that strategies are information dependent by being uniform over cells in the knowledge partition. Global strategies for agents are lifted in global strategy-profiles by taking the Cartesian product of individual strategies, in the function starting at line 146. The pair of announcements corresponding to each strategy profile is computed at line 151 by filtering relevant formulae for each equivalence class. This gives rise to a joint announcement formula obtained by taking the conjunction of individual announcements, in line 156.

```
announcementsKK :: (Eq state, Ord state, Num state) =>
  EpistM state -> state -> [[Form]]
announcementsKK m s = (map (\y->(map snd ((filter (\x->(elem s (fst x))) (y!!0))
  ++ (filter (\x -> (elem s (fst x))) (y!!1) ))))) (profilesKK m))

jointannouncementKK :: (Eq state, Ord state, Num state) =>
  EpistM state -> [Form]
jointannouncementKK m s = map (\x -> (Conj x)) (announcementsKK m s)
updatesKK :: (Eq state, Ord state, Num state) =>
  EpistM state -> state -> [EpistM state]
updatesKK m s = map (\x -> (upd_pa m x)) (jointannouncementKK m s)
```

```
paynumber :: (Eq state, Ord state, Num state) =>
  EpistM state -> state -> Form -> Integer
paynumber m s f | isTrueAtMaybe m s f == Just True = 1
  | otherwise = 0
lpayKK ::(Eq state,Ord state,Num state)=>EpistM state->state->Form->[Integer]
lpayKK m s f = map (\x -> (paynumber x s f)) (updatesKK m s)
```
Chapter 5. Implementing Questioning Games

5.2 Illustrations using the Implementation

Now that the implementation details have been discussed and explained in detail, we continue by introducing one extended illustrative analysis of a concrete questioning game and discuss some theoretical aspects that emerge.

The illustration below follow the code description and explanation. The output will stay very close to the succession of processing functions in Haskell implementation for games with questioning moves previously discussed. Whenever needed the names of the functions used in the output can be used refer back to code sections for further details, documentation and further explanations.

5.2.1. Example. [We Are Spies] Consider the following three-state epistemic goal structure where $a$ knows the truth about $p$ and $b$ knows the truth about $q$:

$$\gamma_a = K_b p \rightarrow K_a q \quad \overline{qp} \quad \frac{a}{qp} \quad \frac{b}{qp} \quad \overline{qp} \quad \gamma_b = K_a \neg q$$

We start by considering the epistemic model used in the ‘We Are Spies’ example (henceforth WaS). The starting epistemic structure is:
5.2. Illustrations using the Implementation

DELq formulae can be evaluated in such structures. Formulas are represented and visualized in Haskell in a compact way, see Section 3.2 for some intuitive examples. Another important element in modeling QAGs are the dynamic epistemic actions changing the underlying epistemic structures. Such epistemic actions can be questions or answers and they are represented as model transformations.

For instance, the result of updating the epistemic model considered in Example 1 with the formula $K_a p$ is computed in and visualized by QAGames.lhs as follows:

```
*QAGames> displayS5 (upd_pa m79 (K a p))
[1,3]
[(1,[p]),(3,[p,q])]
(a,[[1,3],[]])
(b,[[1],[3]])
[1,3]
```

Finally, the truth-value of formulas can be computed at a state in a model. The following is an illustration of how QAGames.lhs performs model checking:

```
*QAGames> isTrueAtMayb (upd_pa m79 (K a p)) 1 (K b q)
False
```

This says intuitively that in the model of Example 5.2.1 updated by announcing $K_a p$ the formula $K_b q$ does not hold at the world with index 1.

5.2.1 Counting Strategies

There are two ways to consider a strategy in a question-answer game. One is the syntactic way, not very convenient because makes the number of possible moves in the game infinite as $\varphi$, $\varphi \land \varphi$, etc. are syntactically different strategies. If not for conceptual reasons, at least as far as an efficient implementation is considered desirable, it would be useful to restrict this number.

The second, semantic, perspective can be used to reduce this number: two strategies are equivalent if they have the same extension. This is the notion of logical equivalence between formulas. Even so, for every subset of the domain there are infinitely many formulae that have that extension, but for a finite model we will have only a finite set of subsets. It is enough to take one representative of each equivalence class. For practical reasons we will use the minimal representative.

When using a semantic approach for strategy equivalence and syntactic entities in a language to represent the moves in the game is very useful to establish a meaningful correspondence between the two levels. This can be done by naming a model, that is adding a nominal to each world’s valuation list. A nominal is a propositional atom which is true in only one world in the domain:
Chapter 5. Implementing Questioning Games

This can be then used to construct the set of all extensionally non-equivalent formulae expressible in the language with respect to a given epistemic model.

In the (WaS) example we have as many extensionally non-equivalent formulae:

\[\{v[], v[n2], v[n1], v[n1,n2], v[n], v[n,n2], v[n,n1], v[n,n1,n2]\}\]

Such formulae are disjunction of propositional letters true at only one world, or nominals, which correspond directly to subsets of the domain after the worlds in the initial epistemic model have been named.

As discussed before, using nominals provides extra expressive power at no computational expense. If, however, the model we are working with is minimized under bisimulation, nominals are not needed, because a characteristic formula of epistemic logic could be used instead to identify any world. Even so, we will see that in the context of associating payoffs to worlds the number of worlds satisfying the same formulae matters and minimization under bisimulation erases relevant information for computing the payoffs in a question-answer game.

However, our example only has local games with two, not eight, strategies. This is because some of the formulae have equivalent semantic effects in the game. This proliferation, however, turns out to be redundant in the sense that it can be avoided using a more efficient compact representation without affecting solution concepts. For this reasons, our reduction gives rise to a natural and economic criterion of strategy equivalence in question answer games. This basic fact is captured by the following more general result for equilibria with pure strategies:

5.2.2. FACT. Let \(M\) be a named epistemic model, \(\Pi = \varphi(\text{dom } M)\) be the power-set of \(M\)'s domain. \([X]_M\) be the set of all formulae equiextensional with \(X \in \Pi\). \(\Phi = \{[X]_M \mid X \in \Pi\}\) be the set of all co-extensional formulas expressible in \(M\), \(||X||_M = \{S_0, S_1, S_2\}\) be the execution-value of a formula with extension \(X\), where \(S_0 = \{w \in M \mid M \models w K_a \neg \varphi\}\), \(S_1 = \{w \in M \mid M \models w K_a \varphi\}\), \(S_2 = \{w \in M \mid M \models w \neg(K_a \neg \varphi \lor K_a \neg \varphi)\}\), \(\Gamma = \{||X||_M \mid X \in \Pi\}\) be the set of all formulae which are epistemically equiextensional, \([\varphi]_M\) the epistemical equiextensionality equivalence class of \(\varphi\) in \(M\). Then, tfaeq:

- \(p = (s_i, s_{-i})\) is a NE in \(G(\Phi)\),
- \(p^* = (s^*_i, s_{-i})\) is a NE in \(G(\Gamma)\), for any \(s^*_i \in [s_i]_M\),

where \(G(X)\) is the epistemic game in which player’s strategies belong to \(X\).
Based on this result, for both reasons of conceptual simplicity and computational convenience we will adopt this further simplification. This will provide a principled definition for a strategy in an epistemic game with question moves which is conceptually clear and computationally efficient.

For instance, in (WaS) asking \( q \) or \( p \) would both have identical effects. And they are both globally different from the effect of asking the trivial question \( \top \):

\[
(v[n1,n2],[[2],[1,3],[\emptyset]]) \quad --\text{the first element represents } q\text{'s extension}
\]
\[
(v[n,n2],[[1,3],[\emptyset],[2]]) \quad --\text{the second element is } p\text{'s execution value}
\]
\[
(v[n1,n1,n2],[[1,2,3],[\emptyset],[\emptyset]]) \quad --\text{Top's exec. value differs from both } p\text{'s & } q\text{'s}
\]

We use this basic observation as a notion of strategy equivalence for questions. When computing the strategy set in the global game we only take the minimal representatives in each equivalence class so determined.

Strategy profiles in the global game are the cartesian product of global strategies and choices in global strategy profiles are uniform across agents’ information cells like in games with imperfect information. The global strategy profiles in the induced game of (WaS) example are given by:

\[
\text{*QAGames> display 1 (profGlob (named m79))}
\]
\[
\text{[[([1],v[]), ([2],v[])], ([1],v[]), ([2,3],v[])], }\]
\[
\text{[([1],v[]), ([2],v[])], ([1],v[n2]), ([2,3],v[])], }\]
\[
\text{[([1],v[]), ([2],v[])], ([1],v[n2]), ([2,3],v[n2])], }\]
\[
\text{[([1],v[]), ([2],v[n2])], ([1],v[]), ([2,3],v[])], }\]
\[
\text{[([1],v[]), ([2],v[n2])], ([1],v[n2]), ([2,3],v[])], }\]
\[
\text{[([1],v[]), ([2],v[n2])], ([1],v[n2]), ([2,3],v[n2])], }\]
\[
\text{[([1],v[n2]), ([2],v[])], ([1],v[]), ([2,3],v[])], }\]
\[
\text{[([1],v[n2]), ([2],v[])], ([1],v[n2]), ([2,3],v[n2])], }\]
\[
\text{[([1],v[n2]), ([2],v[n2])], ([1],v[]), ([2,3],v[])], }\]
\[
\text{[([1],v[n2]), ([2],v[n2])], ([1],v[n2]), ([2,3],v[n2])], }\]
\[
\text{[([1],v[n2]), ([2],v[n2])], ([1],v[n2]), ([2,3],v[n2])], }\]
\[
\text{[([1],v[n2]), ([2],v[n2])], ([1],v[n2]), ([2,3],v[n2])], }\]

A brief inspection of the set of global strategy profiles confirms the fact that there are indeed only two strategies with distinct execution value in the game model. The strategies correspond to disjunction of nominals that are minimal in their equivalence classes. In our example this are the disjunction of the empty list and a disjunction with only one element \( n2 \).

Having a clear cut concept of strategy equivalence will serve as the starting point in computing the outcomes and the construction of the game matrix.

**5.2.2 Computing the Outcomes**

The next lists contain the answers induced by the global profiles at states 1,2,3:
Answers are epistemic formulae about strategies as nominal disjunctions:

Answers to questions by agent $a$ are formulae describing agent $a$’s epistemic state with regard to the content of the question, in a tripartite execution value.
5.2. Illustrations using the Implementation

\[
[b] - v[1], -v[a]v[n2], [b] - v[n2], [a] - v[1]
\]

\[
-b[v[b]v[n2], [b] - v[n2]], [a] - v[1]
\]

\[
-b[v[b]v[n2], [b] - v[n2]], -v[a]v[n2], [a] - v[1]
\]

\[
-b[v[b]v[n2], [b] - v[n2]], -v[a]v[n2], [a] - v[1]
\]

\[
-b[v[b]v[n2], [b] - v[n2]], [a] - v[1]
\]

\[
-b[v[b]v[n2], [b] - v[n2]], -v[a]v[n2], [a] - v[1]
\]

The next lists contain all the updated models in example WaS at worlds 1,2,3:

*QAGames> display 1 (map showS5 (w2updates (named m79) 1))
[[1,2,3], [[(1, [p, n]), (2, [q, n1]), (3, [p, q, n2])], (a, [[1, 3], [2]]), (b, [[1], [2, 3]]), [1, 2, 3]]
[[1,2,3], [[(1, [p, n]), (2, [q, n1]), (3, [p, q, n2])], (a, [[1, 3], [2]]), (b, [[1], [2, 3]]), [1, 2, 3]]
[[1,3], [[(1, [p, n]), (3, [p, q, n2])], (a, [[1, 3]]), (b, [[1], [3]]), [1, 3]]
[[1,3], [[(1, [p, n]), (3, [p, q, n2])], (a, [[1, 3]]), (b, [[1], [3]]), [1, 3]]
[[1,2,3], [[(1, [p, n]), (2, [q, n1]), (3, [p, q, n2])], (a, [[1, 3], [2]]), (b, [[1], [2, 3]]), [1, 2, 3]]
[[1,2,3], [[(1, [p, n]), (2, [q, n1]), (3, [p, q, n2])], (a, [[1, 3], [2]]), (b, [[1], [2, 3]]), [1, 2, 3]]
[[1,3], [[(1, [p, n]), (3, [p, q, n2])], (a, [[1, 3]]), (b, [[1], [3]]), [1, 3]]
[[1,3], [[(1, [p, n]), (3, [p, q, n2])], (a, [[1, 3]]), (b, [[1], [3]]), [1, 3]]
[[1], [[(1, [p, n])], (a, [[1]]), (b, [[1]]), [1]]
[[1], [[(1, [p, n])], (a, [[1]]), (b, [[1]]), [1]]
[[1], [[(1, [p, n])], (a, [[1]]), (b, [[1]]), [1]]
[[1], [[(1, [p, n])], (a, [[1]]), (b, [[1]]), [1]]
[[1], [[(1, [p, n])], (a, [[1]]), (b, [[1]]), [1]]
[[1], [[(1, [p, n])], (a, [[1]]), (b, [[1]]), [1]]
[[1], [[(1, [p, n])], (a, [[1]]), (b, [[1]]), [1]]

These correspond to all the possible executions of strategy values as joint public announcements of the players’ formulae.

These also represent all the game evolution histories and the resulting epistemic structures in which the goal formulae are going to be model checked to determine the payoffs and the overall result of the game.

The difference between the updates at worlds in the domain is determined by the different execution values for the strategies at the worlds.

*QAGames> display 1 (map showS5 (w2updates (named m79) 2))
[[1,2,3], [[(1, [p, n]), (2, [q, n1]), (3, [p, q, n2])], (a, [[1, 3], [2]]), (b, [[1], [2, 3]]), [1, 2, 3]]
[[2], [[(2, [q, n1])], (a, [[2]]), (b, [[2]]), [2]]
[[1,2,3], [[(1, [p, n]), (2, [q, n1]), (3, [p, q, n2])], (a, [[1, 3], [2]]), (b, [[1], [2, 3]]), [1, 2, 3]]
[[2], [[(2, [q, n1])], (a, [[2]]), (b, [[2]]), [2]]
[[2,3], [[(2, [q, n1]), (3, [p, q, n2])], (a, [[2], [3]]), (b, [[2, 3]]), [2, 3]]
[[2], [[(2, [q, n1])], (a, [[2]]), (b, [[2]]), [2]]
[[2,3], [[(2, [q, n1]), (3, [p, q, n2])], (a, [[2], [3]]), (b, [[2, 3]]), [2, 3]]
[[2], [[(2, [q, n1])], (a, [[2]]), (b, [[2]]), [2]]
[[1,2,3], [[(1, [p, n]), (2, [q, n1]), (3, [p, q, n2])], (a, [[1, 3], [2]]), (b, [[1], [2, 3]]), [1, 2, 3]]
[[2], [[(2, [q, n1])], (a, [[2]]), (b, [[2]]), [2]]
[[2,3], [[(2, [q, n1]), (3, [p, q, n2])], (a, [[2], [3]]), (b, [[2, 3]]), [2, 3]]
[[2], [[(2, [q, n1])], (a, [[2]]), (b, [[2]]), [2]]
Chapter 5. Implementing Questioning Games

It is easy to check that the resulting updated models represented here as code output correspond to the resulting models from Example 5.2.1.

The following are the payoffs obtained by player \(a\) in the game of Example WaS:

\[
\text{QAGames> display 1 (zip3 (w2pays (named m79) 1
(\text{imp (Disj}[K b p, K b (Neg p)]) (Disj}[K a p, K a (Neg p)])))
(w2pays (named m79) 2
(\text{imp (Disj}[K b p, K b (Neg p)]) (Disj}[K a p, K a (Neg p)])))
(w2pays (named m79) 3
(\text{imp (Disj}[K b p, K b (Neg p)]) (Disj}[K a p, K a (Neg p)]))
(1,1,1) (1,1,1)
(1,1,1) (1,1,1)
(1,1,1) (1,1,1)
(1,1,1) (1,1,1)
(1,1,1) (1,1,1)
(1,1,1) (1,1,1)
(1,1,1) (1,1,1)
(1,1,1) (1,1,1)
\]

The following are the payoffs obtained by player \(b\) in the game of Example WaS:

\[
\text{QAGames> display 1 (zip3 (w2pays (named m79) 1
(\text{imp (Disj}[K b p, K b (Neg p)]) (Disj}[K a p, K a (Neg p)])))
(w2pays (named m79) 2
(\text{imp (Disj}[K b p, K b (Neg p)]) (Disj}[K a p, K a (Neg p)])))
(w2pays (named m79) 3
(\text{imp (Disj}[K b p, K b (Neg p)]) (Disj}[K a p, K a (Neg p)]))
(1,0,0) (1,0,0)
(1,1,1) (1,1,1)
(1,0,0) (1,0,0)
(1,1,1) (1,1,1)
\]
The payoff obtained by the players is constructed by averaging the local result of checking goal formulae over the entire domain. For ease of readability we will only take here the sum without dividing it to the size of the domain.

The next output gives the game matrix for the Q-A game in the WaS example:

\[
\begin{array}{cccc}
(1,0,0) & (1,0,0) \\
(1,1,1) & (1,1,1) \\
(1,0,0) & (1,0,0) \\
(1,1,1) & (1,1,1)
\end{array}
\]

5.2.3 Counting Goals

There are sixteen global strategy profiles in the game from the WaS example:

\[
\text{display 4 (qagn (named m79))}
\]

...
This extended structure captures all the needed information about the game. As far as the truth value of epistemic formulae is concerned many of these worlds are equivalent.

Using the standard epistemic functionality in our code we can minimize this model under bisimulation and name it. This minimization process is desirable and useful especially for very large instances, however, it also has some drawbacks for our current purpose, which is more then just preserving truth of the formulae.
Next, we show how one can compute the minimal model more efficiently using partitions to capture the tree structure of the game history. Let $W$ be the domain of a model $M = \langle W, R, V \rangle$ for $a \in A$. Let $L_W$ be the lattice of partitions of $W$ with $\wedge$ refinement and $\vee$ coarsening. Let $P_a$ be agent $a$’s information partition. Let $L_W | P_a$ be the $a$-conditional partition lattice of $W$. Then

$$\min(M) = \bigcup_{a \in A} (L_W | P_a) \cup \bigvee_{a \in A} \{L_W | P_a\}$$

This confirms that there are as many logically non-equivalent goal formulae:

This is an efficient and convenient model transformation that preserves the truth value of all modal formulae. However, this also erases relevant information about the strategic aspects of the epistemic game. For instance, there are as many worlds that satisfy the formula: $p \land \neg q \land K_a K_b K_a \neg p$:

Whereas, after the model is minimized under bisimulation it appears as if the goal is satisfied only once. This will affect the computation of the final payoffs in the game whenever the model is minimized under standard bisimulation.

The same happens for the following goal formulas satisfied twelve times:

The following formulae are each satisfied eight times in the following worlds:
Chapter 5. Implementing Questioning Games

And the following formulae are satisfied each in the listed worlds, respectively:

To avoid this information loss we can use a hoarding version of bisimulation contraction, to keep track of how many worlds we have in each contracted equivalence class across the model. This can be represented by encoding the size of the bisimilarity equivalence class as a parameter in the minimal model. A refined criterion of goal equivalence should also keep track of this amount. See the notion of probabilistic bisimulation from Chapter 8 for a comparison.

Considering a strictly extensional criterion, and no restriction with regard to the expressive power of the language or the syntactic structure of the formulas expressing goals we have so many logically equivalent goal formulae:

As this example already illustrates, even the analysis of the simplest scenarios can lead to structures with a very large number of states. This is even more so for the setting using product update for more complex questioning and resolution actions like the ones we discussed in the previous chapter. This is usually called the ‘state explosion’ problem and to avoid it is desirable to have algorithms for minimizing structures while preserving behavioral equivalence.
5.3 Birelational Coarsest Partition Problem

The main distinctive feature in our approach to questions so far was the use of the intersection between two accessibility relations. This has technical consequences with regard to behavioral invariance for issue-epistemic models that were already explained and discussed in previous chapters. The main idea is that the intersection modality is not invariant under bisimulation.

However, the ability to work with models that capture the same structural behavior with a minimal number of states is very important in practice and also desirable from a theoretical point of view. This is the problem of computing for a given model the minimal bisimilar model.

For epistemic models this problem is solved using the standard notion of bisimulation by a standard partition refinement algorithm. This is a problem also related to minimizing the number of states in a finite automaton, see [74] and [109] for further details.

For issue-epistemic models we cannot just use the standard minimization under bisimulation procedure for the already mentioned reason that this does not adequately capture the behavior of the intersection modality.

In this section we define the Birelational Coarsest Partition Problem in similar way the coarsest partition problem is defined in [74] and we give an algorithm that solves it similar to the algorithm from [109]. The main advantage of having such an algorithm for issue-epistemic structures is that it provides a way to capture invariance of issue-epistemic formulae, including, in special, formulae containing the intersection modality which are going to be also preserved by this algorithm.

5.3.1. DEFINITION. [Bistability] Let $W$ be a finite set and $K \subseteq W \times W$ and $Q \subseteq W \times W$ be two binary relations over $W$. A set $B \subseteq W$ is bistable with respect to another set $D \subseteq W$ if either $B \subseteq (Q \cap K)(D)$ or $B \cap (Q \cap K)(D) = \emptyset$.

A partition $P$ of $W$ is bistable with respect to a set $D \subseteq W$ if all the blocks belonging to $P$ are bistable as sets with respect to $D$. $P$ is self-bistable if it is bistable with respect to each of its own blocks as sets.

Two consequences of bistability, defined as in [74] useful during later proofs:

1. Bistability is inherited under refinement; that is, if $Q$ is a refinement of $P$ and $P$ is bistable with respect to a set $S$, then so is $Q$.
2. Bistability is inherited under union; that is, if $P$ is bistable with respect to two sets $Q$ and $S$, then $P$ is also stable with respect to $Q \cup S$.

The birelational coarser partition problem is that of finding for two given relations $K$ and $Q$ and initial partition $P$ over a set $W$ the coarsest stable refinement of $W$, i.e., the partition such that every other stable partition is a refinement of it, so that it has the fewest blocks. Lemma 5.3.4 will also show that the coarsest stable refinement is unique.

For any partition $Q$ and subset $S \subseteq W$, let $\text{split}(S_R, Q)$ be the refinement of $Q$ obtained by replacing each block $B \in Q$ such that $B \cap R(S) \neq \emptyset$ and $B \setminus R(S) \neq \emptyset$
by the two blocks $B' = B \cap R(S)$ and $B'' = B \setminus R(S)$. The set $S$ is a splitter of $Q$ if $\text{split}(S_R, Q) \neq Q$.

5.3.2. Definition. [Double Split] For any partition $Q$ and subset $S \subseteq W$, let $\text{split}(S_2, Q)$ be the refinement of $Q$ obtained by replacing each block $B \in Q$ such that $B \cap (Q \cap K)(S) \neq \emptyset$ and $B \setminus (Q \cap K)(S) \neq \emptyset$ by the two blocks $B' = B \cap (Q \cap K)(S)$ and $B'' = B \setminus (Q \cap K)(S)$.

The set $S$ is a double splitter of $Q$ if $\text{split}(S_2, Q) \neq Q$.

Note that $Q$ is unstable with respect to $S$ if and only if $S$ is a splitter of $Q$. The same for bistable and double split. Two properties of the $\text{split}$ function, defined as in [74], that are going to be used in later proofs are:
1. Function $\text{split}$ is monotone in the second argument; that is, if $U \subseteq W$ and $P$ is a refinement of $Q$ then $\text{split}(U, P)$ is a refinement of $\text{split}(U, Q)$.
2. Function $\text{split}$ is commutative. The coarsest partition of $P$ bistable with respect to both $S$ and $Q$ is $\text{split}(S, \text{split}(Q, P)) = \text{split}(Q, \text{split}(S, P))$.

<table>
<thead>
<tr>
<th>Algorithm 1 Compute Coarsest Birelational Stable Refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition:</strong> $Q$ is a partition of $W$, $R_1$, $R_2$ are binary relations on $W$</td>
</tr>
<tr>
<td><strong>Postcondition:</strong> $Q$ is the coarsest bistable refinement stable for $R_1$ and $R_2$</td>
</tr>
<tr>
<td>repeat</td>
</tr>
<tr>
<td>1. Find a set $S$ that is a union of $Q$-blocks and is a double-splitter of $Q$</td>
</tr>
<tr>
<td>2. Replace $Q$ by $\text{split}(S, Q)$</td>
</tr>
<tr>
<td>3. Find a set $S$ that is a union of $Q$-blocks and is a $R_1$-splitter of $Q$</td>
</tr>
<tr>
<td>4. Replace $Q$ by $\text{split}(S, Q)$</td>
</tr>
<tr>
<td>5. Find a set $S$ that is a union of $Q$-blocks and is a $R_2$-splitter of $Q$</td>
</tr>
<tr>
<td>6. Replace $Q$ by $\text{split}(S, Q)$</td>
</tr>
<tr>
<td>7. until $Q$ is self-bistable and stable with regard to $R_1$ and $R_2$</td>
</tr>
</tbody>
</table>

5.3.3. Lemma (Invariance). Algorithm 1 maintains the invariant that every coarsest stable refinement of the initial partition $P$ is also a refinement of the current partition $Q$.

The proof is very similar to the proof in [74], it proceeds by induction on the number of refinement steps using the four properties discussed so far. The new contribution is the use of the previously described split function for two relations.

5.3.1. Proof (Lemma 5.3.3). By induction on the number of refinement steps. The base case is established by definition. Suppose we have proved that the invariant is maintained before a refinement step of a partition $Q$ by a splitter set $S$. If $R$ is an arbitrary coarsest stable refinement of $P$. Because $S$ is a union of $Q$-blocks and $R$ is a refinement of $Q$ by IH, $S$ is a union of blocks of $R$. Therefore, $R$ is bistable with respect to $S$. Because the $\text{split}$ function is monotone, $R = \text{split}(S, R)$ is a refinement of $\text{split}(S, R)$. □
5.3.4. **Lemma (Termination).** The refinement process in Algorithm 1 is correct and terminates after at most $n-1$ refinement steps, having computed the unique coarsest bistable partition.

The proof is very similar to the proof in [74], it is based on the fact that the number of blocks in $Q$ is at most $|W| = n$.

5.3.2. **Proof (Lemma 5.3.4).** Because the number of partition cells in $Q$ for a finite domain is at least one and at most $|W| = n$ and because each refinement step either increases this number or has already reached the least fixed point, the algorithm terminates after $n-1$ refinement steps. After the step in which no further refinement steps are possible, $Q$ is bistable and by Lemma 5.3.3 any bistable refinement is a refinement of $Q$. Hence $Q$ is the unique coarsest bistable refinement.

We will use Algorithm 1 as the core component inside a model minimization process that preserves modal formulae in the questioning language, including formulae containing intersection modalities. This will be very similar with the iterative refinement process from Algorithm 1 but for which some extra processing has to be performed in order to make it adequate for the propositional and local structure of issue-epistemic models.

There are two main aspects that need additional careful consideration: the first one concerns the starting state of the algorithm, the second one concerns the final steps and the resulting model.

At the beginning of the algorithm, before the refinement process starts there is some preprocessing needed i.e. we have to make sure that the initial partition is not an arbitrary one but one that respects atomic harmony, and is therefore a partition in propositionally equivalent blocks.

After the refinement process reaches a fixed point we still need to apply some postprocessing steps in order to build a new issue epistemic model from the existing structure of the partition blocks and the relational structure in the initial issue-epistemic model.

All we have to do before we can prove correctness and termination for our algorithm is to spell out all the details in the build function, that is describe the way in which the final model is constructed.

Intuitively, we take the new domain to be the set of partition blocks, the new issue relation is built from the initial one as follows:

$$[x] \approx [y] \iff \forall z \in [x] \exists v \in [y] : z \approx v$$

The new epistemic relation is constructed from the initial one in an analogous manner. The valuation of each minimal representative of a partition block is assigned to the block containing it. And the issue of actuality is solved via block membership projection.
Algorithm 2 Compute The Minimal Questioning Model

**Precondition:** $M$ is an arbitrary Issue-epistemic Model

**Postcondition:** $M'$ is the minimal PIM intersimilar to $M$

1. $P \leftarrow \{ C_i \mid C_i \subseteq \text{dom}(M), \forall v, w \in C_i : V(w) = V(v) \}$
2. repeat
   3. Find a set $S$ that is a K-splitter of $P$ and a union of $P$ blocks
   4. $P \leftarrow \text{split}(S, P)$
   5. Find a set $S$ that is a Q-splitter of $P$ and a union of $P$ blocks
   6. $P \leftarrow \text{split}(S, P)$
   7. Find a set $S$ that is a 2-splitter of $P$ and a union of $P$ blocks
   8. $P \leftarrow \text{split}(S, P)$
3. until $P$ is self-bistable and stable with regard to Q and K
4. $M' \leftarrow \text{build}(M, P)$

We can now show that modal formula in the questioning language are preserved during the process of model minimization:

**5.3.5. Lemma (Preservation).** Algorithm 2 ensures that questioning modal formulae true in the initial model are also true in the minimal model, including, in special, formulae using intersection modalities.

The proof proceeds by induction on the structure of a modal formula using the previous definitions for the split and double-split functions used in the refinement steps and the construction of the new model via the build function.

**5.3.3. Proof (Lemma 5.3.5).** Let $M$ be an arbitrary probabilistic issue epistemic model. Let $M^-$ be constructed from $M$ via Algorithm 3.

We will proceed by constructing a double splitting set from any formula that changes its truth value from $M$ to $M^-$. Let $\varphi$ be a diffraction formula, i.e. let $M \models_w \varphi$ and $M^- \not\models_{[w]} \varphi$.

We continue by induction on the structure of the formula $\varphi$.

If $\varphi$ is an atomic propositional formula then we have from the definition of the build function that $\varphi \in V^-([w])$ and we are done.

If $\varphi$ is a negation or a conjunction we apply IH to the constituent formula (e).

If $\varphi := \langle \approx \cap \neg \rangle \psi$ then, from $M \models_w \langle \approx \cap \neg \rangle \psi$ we obtain by modal semantics that $\exists x \in W : w \approx \cap \neg x$ and $M \models_x \psi$. Using the induction hypothesis we get that $M^- \models_{[x]} \psi$.

On the other hand, from $M^- \not\models_{[w]} \varphi$ we obtain that $\forall [x] \in W^- : [w] \approx \cap \neg [x]$ implies $M^- \not\models_{[x]} \psi$. Therefore, it must be the case that $([w], [x]) \not\in \approx \cap \neg$. From the definition of the build function we have that $\exists v \in [w]$ and $\exists y \in [x]$ such that $(v, y) \not\in \approx \cap \neg$.

Assume wlog that $v \not\approx y$ then we can easily check that if we take the set $S = [w]$ we have that $\text{split}(S, P) \neq P$. Hence $S$ is a double splitter set of $P$. 
and $P$ could not have been obtained as the fixed point of the refinement process because is not self-bistable.

For the remaining cases of formulae using the epistemic and questioning modalities $Q$ and $K$ the argument proceeds completely analogously using $\approx$ and $\sim$ and constructing a splitter set.

5.3.6. Lemma (Termination). The refinement process in Algorithm 2 is correct for behavioral equivalence and terminates after at most $|W| + 1$ refinement steps, having computed the unique minimal model intersimilar to the initial model.

The proof is based on the fact that we always start from an issue-epistemic structure with a finite domain. Therefore the number of blocks in the initial propositional equivalence partition $Q$ is at most $|W| = n$.

5.3.4. Proof (Lemma 5.3.6). Because the starting propositional equivalence partition in a finite issue-epistemic structure has at most as many cells as the size of the domain of the model, the number of partition cells in $P$ is at least one and at most $|W| = n$. Each time the repeat loop is executed the number of blocks in the partition is increased, therefore in at most $n + 1$ steps the repeat cycle will bistabilize the partition and by Lemma 5.3.3 no further refinement is possible, so no smaller model can exist. By Lemma 5.3.5 the model built is intersimilar with the initial one. Hence $M'$ is the unique minimal model intersimilar to $M$ and therefore behaviorally equivalent.

To conclude, we have provided an algorithm that can be used to minimize issue-epistemic models while preserving the truth value of formulas using intersection modalities. This will also be very useful for minimizing probabilistic issue-epistemic models later on in the chapter.

The insights provided by the minimization algorithm can be made more precise and can be generalized in a structural notion describing invariance for the issue-epistemic language. For this we have to come up with an adequate notion of behavioral equivalence for issue-epistemic structures. In order to achieve this we will need a notion of invariance between issue-epistemic models. This notion cannot be standard bisimulation because, as we discussed before, the formulae containing intersection modalities are not preserved under the standard epistemic bisimulation. Hence the first step will be to propose an improved version of behavioral equivalence. We will need a notion that is adequate in the following sense: it will preserve the truth value of all formulae needed to reason about questioning scenarios, in special formulae using the interdependence between the two relations issue and epistemic, but also those using only one.

We will call this notion intersection bisimulation or intersimulation for short, to highlight the main feature that it is designed to address.
5.3.7. Definition. [Intersection Bisimulation] An intersection bisimulation between two IEMs $M$ and $M'$ is a relation $Z \subseteq W \times W'$ defined as in Definition 6.5.4 but in which the first two clauses describing the probabilistic aspects are ignored. So an intersimulation has the following properties:

Atomic harmony: if $sZs'$ then $V(s) = V'(s')$

Forward relations: for both relations: $sZs'$ and $sRt$ for some $t \in W$ implies that there is some $t' \in W'$ with $s'R't'$ and $tZt'$

Forward intersection: if $sZs'$ and both $sRt$ and $sSt$ hold for some $t \in W$ then there is some $t' \in W'$ with both $s'R't'$ and $s'S't'$ and $tZt'$

Backward relations: symmetrical with the forward clause for relations

Backward intersection: symmetrical with the forward clause for intersection

We will now argue that the notion of intersimulation is the adequate invariance notion for a language with intersection modalities:

5.3.8. Theorem (Invariance). For any two pointed PIMs $M$ and $M'$ and for any formula $\varphi$ in a questioning language with intersection modalities we have:

if $M, w \leftrightarrow M', w'$ then $M \models w \varphi$ iff $M' \models w' \varphi$.

The proof proceeds by induction on the structure of the formula $\varphi$.

5.3.5. Proof (Theorem 5.3.8). Atomic and Boolean cases follow straightforwardly from the atomic clause and IH. For the modal formulae: In case $\varphi := \mathcal{R}\tilde{\psi}$, suppose $xZy$ and $M \models x \varphi$. From $xZx'$ we get using Definition 6.5.4 that $\mathcal{R}(QKW_x, Q'K'W'_y)$ which is a shortcut notation for the following condition: $\mathcal{R} \{(v \in W \mid xQvKx), \{v' \in W' \mid yQ'v'K'y\}\}$. This is equivalent with: $\forall v \in \{v \in W \mid xQvKx\} : \exists v' \in W' \mid yQ'v'K'y : vZv'$, which is further translated into $\forall v \in \{v \in W \mid (x, v) \in Q \land (v, x) \in K^{-1}\} : \exists v' \in \{v' \in W' \mid (y, v') \in Q' \land (v', y) \in K'^{-1}\} : vZv'$. By IH we obtain that $M \models v \psi$ iff $M' \models v' \psi$. We also have $\forall v \in \{v \in W \mid (x, v) \in Q \land K\} : \exists v' \in \{v' \in W' \mid (y, v') \in Q' \land K'\} : vZv'$. Hence we can conclude, as desired, that $M' \models v' \varphi$. The other direction is similar. The cases for the remaining modalities are completely analogous using the corresponding relation and the matching clause in Definition 6.5.4. □

We will show in Section 6.5 that this notion can also stand as the first building block for a notion of probabilistic intersection bisimulation.