Dynamic knowledge logic

Hans van Ditmarsch*

March 8, 2000

Abstract

Modelling the epistemic dynamics of multiagent systems currently gets much attention from the research community, see e. g. [Ger99], [FHMV95], [BMS98], [Koo99], [Ren99]. Contrary to most approaches, we restrict ourselves to $S5$ models, see [vD99]. We propose a language KT of knowledge action types and a derived language KA of knowledge actions. Basic to our approach is the concept of local interpretation, where 'local' means: only for the agents involved in that action. The language can be fruitfully applied to describe actions in knowledge games.

1 Introduction

In [vD00d], we described what knowledge games are. In [vD00a], we argue that knowledge game states can be seen as pointed multiagent $S5$ models, i.e. as multiagent $S5$ states. We now proceed by defining a programming language for game actions in knowledge games, such as showing a card to another player. An action tranforms one game state into another one. A program corresponding to such an action should therefore be interpreted as a functional binary relation between pointed multiagent $S5$ models, corresponding to game states. Our definitions will be more general in two respects: first, we forget about knowledge *games*, and just consider operations transforming multiagent $S5$ states. These will be called *knowledge actions*. Second, more basic than an action is a type of action. A *knowledge action type* is an operation transforming a multiagent $S5$ *model*, i.e. *without* designated point, into another (or several other) model(s). Just as a model can be seen as a set of states, a knowledge action type can be seen as a set of *knowledge actions*. Because examples of knowledge actions are actions in games, we start by a review of some actions in a standard knowledge game situation.

Three players 1, 2 and 3 each hold one of three cards red, white and blue. Players can only see their own card. This is all common knowledge. The actual

*University of Groningen, Cognitive Science and Engineering, Grote Kruisstraat 2/1, 9712 TS Groningen, the Netherlands, hans@cs.rug.nl, http://tcw2.ppsw.rug.nl/~hans/

dealing of cards is: 1 holds red, 2 holds white, and 3 holds blue. Some actions in this game state are:

- **example 1**: player 1 puts the red card (face up) on the table
- **example 2**: player 1 shows (only) player 2 the red card (so that player 3 can't see the shown card)
- **example 3**: player 2 asks player 1 for the white card; player 1 says that he doesn't have it; player 2 ends his move, i.e. makes public that he doesn't know the dealing of cards
- **example 4**: player 2 asks player 1 to tell him a card that he (1) doesn't have; player 1 whispers in 2's ear that he doesn't have the white card (so that player 3 can't hear what is being said)

All these actions can be described in our proposed action language KA, and their interpretation can be computed on the initial game state. Other, seemingly more complicated actions can be described and interpreted as well, such as (it is commonly known that) '1 suspects 2 to have peeked into his card'. We do not aim to describe actions that disturb the $S5$ character of game states, such as private communications, non-public suspicions, etc. Still, other sorts of action that also result in $S5$ models will appear to be outside the reach of our language. Example:

Same as the previous example, only we add one more card, and we rename the cards to: north, east, south and west. Player 3 holds two cards. The actual dealing of cards is: 1 holds north, 2 holds east, and 3 holds south and west. Some actions in this game state are:

- **example 5**: player 3 shows his south card (only) to player 1, with his left hand, and (simultaneously) his west card (only) to player 2, with his right hand
- **example 6**: player 3 shows his south card (only) to player 1; next, player 2 asks player 3 to show him his other card; player 3 shows his west card (only) to player 2

Now that we have an idea of the actions we want to model, we discuss the program constructing operators that we need in our action language. We let ourselves be guided by the game actions that we want to model, by the operators available in a dynamic logic as in [Har84] and [Gol92], and by the update operation in [Ger99]. We propose the following six program constructing operators: test, sequential execution, nondeterministic choice, simultaneous (parallel) execution, learning, and 'local choice'. We motivate these operators by the examples given above.

We want to *test* propositions describing whether a person holds a card, or whether a player, or a group of players, know(s) or does (do) not know something. As a game can consist of a chain of actions, as in example 3, we need *sequential execution*. In order to describe all the actions that are available to a player given a certain game state, we need *nondeterministic choice*. In example 4, above, where player 1 holds the red card, he can choose between telling 2 that he doesn't have white, or telling him that he doesn't have blue.

If a player shows a card to another player, they both have 'learnt', so to speak, that the first player holds that card. In example 1, all three players have learnt that player 1 holds the red card. In example 2, players 1 and 2 have learnt that player 1 holds the red card. *Learning*, as we define it, is much related to 'updating', as in [Ger99]. There is an important difference: one can only learn things that are true (or, more properly and generally put: programs that are executable), whereas one can update with a proposition regardless of its truth or falsity.

The operation of *local choice* is needed to describe actions where different subgroups learn different things. Typical is the action where a player is showing a card to (only) another player, as in example 2. In that example, the subgroup consisting of 1 and 2 learns that 1 holds the red card, whereas the group consisting of 1, 2 and 3 learns that 1 and 2 learn which card 1 holds, or, in other words: that either 1 and 2 learn that 1 holds the red card, or that 1 and 2 learn that 1 holds the white card, or that 1 and 2 learn that 1 holds the blue card. One might say that the choice made by subgroup $\{1, 2\}$ from the three alternatives is *local*, i.e. known to them only, because it is hidden from player 3.[1]

Simultaneous execution is needed to describe example 5, the move of a player simultaneously showing cards to different players. Because we do not have an interpretation for it, we will not describe simultaneous execution in the language. Instead, we discuss it in a separate section, 5, at the end.

We also will not be able to describe the action taking place in example 6, but this is for a more general reason: we choose to model actions by their epistemic effects, therefore we *cannot* express when they have been taking place. From the viewpoint of epistemic effects, the action in example 6 would be indistinguishable from: 'player 3 shows his blue card (only) to player 1; three moves later, player 2 asks player 3 to show him his other card; player 3 does so'. This will also be discussed in section 5.

The operations we have described cannot be defined from each other. The role of 'local choice' is rather different from the role of the other operations. The class of programs that we can construct from tests, sequence, choice and learning are the knowledge action types KT. Given a knowledge action type, we can then construct a knowledge action by the operation of local choice. Thus is

[1]The meaning of the word 'choice' is somewhat vague. In 'nondeterministic choice' choice means 'making a choice possible', in local choice it means 'actually choosing'. Of course, 'actually choosing' after 'having made a choice possible' doesn't create a new program but returns an old one. However, 'locally choosing' after having made a choice possible *does* create a new program.

formed the class of knowledge actions KA. Although these classes may be said to overlap (in the deterministic action types), some actions are not types and some types are not actions. We explain this with example 2: '1 shows 2 a card' is a *knowledge action type*, whereas '1 shows 2 his red card' is a *knowledge action* of that type. Action types can be nondeterministic, as here, whereas actions are always deterministic.

The interpretation of a knowledge action type is simultaneously a relation between $S5$ models and between their worlds. Fundamental to this interpretation is the concept of local interpretation, that we need in order to interpret the learning operator. As in 'local choice', local means: 'for a certain subgroup only'. Given the interpretation of a knowledge action type, the interpretation of a knowledge action is determined by choosing one world in the origin of that interpretation, and one world in one of its images.

The class of $S5$ models is closed under application of action types, and the class of $S5$ states is closed under application of actions. This will become obvious from the definitions. In order to guarantee that, we have to rule out programs where *only* a subgroup of agents learns something, because after its execution agents not included in that subgroup erroneously believe that nothing happened at all: that means that the model is no longer reflexive, and therefore no longer $S5$. There are also other constraints on action types.

We now proceed by introducing the language and its intepretation. In section 2 we define the logical language DKL. In section 3 we define the knowledge action types KT and their interpretation. In section 4 we define the knowledge actions KA and their interpretation. In section 5 we discuss possible extensions of the programming language.

In [vD00c] we give an overview of the application of our definitions to knowledge games. In [vD00b] we compare our approach to that of other researchers, in particular [Ger99] and [Bal99].

2 Dynamic knowledge logic

2.1 Logical preliminaries

Knowledge models and knowledge states

We restate some relevant terminology. For a standard introduction, see [MvdH95]. A *knowledge model* is a multiagent $S5$ ($S5_n$) model M. A *knowledge state* is a pointed multiagent $S5$ model (M, w). We also name states with – possibly indexed – lower case letters s, t. If $s = (M, w)$ then s is called a state for model M; M is called the model underlying state s.

In $S5$ we may write $\sim_a$ for the accessibility relation for agent a, as it is an equivalence relation (a partition of the domain). We write $\sim_A = (\bigcup_{a \in A} \sim_a)^*$.

Given a (parameter) set of agents $\mathbf{A}$, we will often relate to models with *only* a partition for agents $a \in A \subset \mathbf{A}$. In that case we say that access is *defined* for agents $a \in A$ and *undefined* for agents $a \in \mathbf{A} \setminus A$. A consequence of this distinction is that, nonstandardly, modal formulas $K_a\varphi$ are undefined, instead of vacuously true, on models with empty access for a, see subsection 2.3.

Group

Let M be a knowledge model, let s be a knowledge state. The *group* of $M = \langle W, (\sim_a)_{a\in A}, V\rangle$ is the set A of agents for which access is defined. When $gr(M) = A$ we say that M is an A *model*, or that M is a *model for group* A. When $gr(s) = A$ we say that s is an A *knowledge state*, or that s is a *knowledge state for group* A. The notion of group plays an important part. We will also define the group of an action type and of an action. We then require that an action type can only be interpreted in a model if their groups are the same, and that an action can only be interpreted in a state if their groups are the same.

2.2 Language of dynamic knowledge logic – DKL

Definition 1 (DKL – Language of Dynamic Knowledge Logic)
Given are a set of atomic propositions $\mathbf{P}$ and a set of agents $\mathbf{A}$. DKL is the smallest set closed under:

$$\begin{array}{rll}
p \in \mathsf{DKL} & \text{if} & p \in \mathbf{P} \\
\neg\varphi \in \mathsf{DKL} & \text{if} & \varphi \in \mathsf{DKL} \\
\varphi \wedge \psi \in \mathsf{DKL} & \text{if} & \varphi, \psi \in \mathsf{DKL} \\
K_a\varphi \in \mathsf{DKL} & \text{if} & a \in \mathbf{A} \text{ and } \varphi \in \mathsf{DKL} \\
C_B\varphi \in \mathsf{DKL} & \text{if} & B \subseteq \mathbf{A} \text{ and } \varphi \in \mathsf{DKL} \\
[\pi]\varphi \in \mathsf{DKL} & \text{if} & \pi \in \mathsf{KT} \cup \mathsf{KA} \text{ and } \varphi \in \mathsf{DKL}
\end{array}$$

KT is the class of knowledge action types, to be defined in the section 3. KA is the class of knowledge actions, to be defined in section 4. 'Program' is the generic term that we use for both types and actions. The parameter set of agents $\mathbf{A}$ is called the *public*. We introduce the usual abbreviations (let $p \in \mathbf{P}$):

$$\begin{array}{rcl}
\varphi \vee \psi & := & \neg(\neg\varphi \wedge \neg\psi) \\
\varphi \rightarrow \psi & := & \neg\varphi \vee \psi \\
\varphi \leftrightarrow \psi & := & (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\
\top & := & p \vee \neg p \\
\bot & := & p \wedge \neg p
\end{array}$$

The '*modal agents of a formula* φ', notation $ag(\varphi)$, is the set of agents occurring in modal operators K_a , C_B and $[\pi]$ in that formula. The operation ag is needed for the interpretation of action types, at the stage where local interpretation comes into play: A test $?\varphi$ can *only* be interpreted in a model M, if $ag(\varphi) \subseteq gr(M)$: if all the modal operators that occur in φ can indeed be evaluated in any world from M. This is a real constraint. Given the public $\mathbf{A}$, and a group $B \subset \mathbf{A}$, M may be a B model, whereas φ may contain operators K_a for agents $a \in \mathbf{A} \setminus B$.

Because formulas can contain dynamic modal operators too, we need to extend ag to action types and actions: the interpretation of a formula with a dynamic modal operator $[\pi]$ depends on the modal agents of the tests occurring in π (and on the agents that learn something in that program). For an inductive definition, see definition 15 in the appendix.

2.3 Semantics of dynamic knowledge logic

Definition 2 (Semantics of dynamic knowledge logic)
Let $M = \langle W, (\sim_a)_{a \in A}, V \rangle$ be a knowledge model, where $A \subseteq \mathbf{A}$. Let w be a world in M. Let $p \in \mathbf{P}$, $a \in \mathbf{A}$, $B \subseteq A$, $\pi \in \mathsf{KT} \cup \mathsf{KA}$. Assume that the agents of the formula to be interpreted are contained in or equal to A.

$$\begin{array}{lll}
M, w \models c & \text{iff} & V_w(c) = 1 \\
M, w \models \neg\varphi & \text{iff} & M, w \not\models \varphi \\
M, w \models \varphi \wedge \psi & \text{if} & M, w \models \varphi \text{ and } M, w \models \psi \\
M, w \models K_a\varphi & \text{iff} & \forall w' : w' \sim_a w \Rightarrow M, w' \models \varphi \\
M, w \models C_B\varphi & \text{iff} & \forall w' : w' \sim_B w \Rightarrow M, w' \models \varphi \\
M, w \models [\pi]\varphi & \text{iff} & \forall M' : \forall w' : (M, w)[\![\pi]\!](M', w') \Rightarrow M', w' \models \varphi
\end{array}$$

A program π can be either an action type $\tau \in \mathsf{KT}$ or an action $\alpha \in \mathsf{KA}$. The interpretation $[\![\tau]\!]$ of an action type is defined in section 3. It is basically a relation between models, from which a relation between states is derived. The interpretation $[\![\alpha]\!]$ of an action is defined in section 4. It is a relation between states.

Let $s = (M, w)$ be a knowledge state. Then $s \models \varphi \Leftrightarrow M, w \models \varphi$.

Warning: in this semantics it is *not* the case that a model without access for a certain agent a satisfies formulas of type $K_a\varphi$ 'ex falso', because there aren't any a-accessible worlds. According to the definition above, the interpretation of such a formula is *undefined* on that model. For a similar reason, we have that $[?K_1p]K_1p$ is always undefined, even on models that satisfy K_1p. This will soon become clear.

3 Knowledge action types

3.1 Language of knowledge action types – KT

We have to choose between either defining a wide class of programs, and give a semantics for only a subclass, or defining this subclass directly, so that we can interpret all programs. We do the last. The inductive definition of the action types KT assumes the simultaneous definition of the *group* $gr(\tau)$ and of the *modal agents* $ag(\tau)$ of an action type τ. The first is the set of agents occurring in learning operators L_B in τ, the second is the set of agents occurring in modal operators in test formulas in τ. See the appendix for their precise definitions.

As tests can again contain programs, we also assume KT and DKL (and KA) to be simultaneously defined.

Definition 3 (KT – **Knowledge action types)**
Given a set of agents $\mathbf{A}$ and a set of atoms $\mathbf{P}$, KT is the smallest set closed under:

3.a	$?\varphi \in \mathsf{KT}$	if	$\varphi \in \mathsf{DKL}$
3.b	$L_B\tau \in \mathsf{KT}$	if	$\tau \in \mathsf{KT}$, and $gr(\tau) \subset B$
3.c	$\tau \ ; \ \tau' \in \mathsf{KT}$	if	$\tau, \tau' \in \mathsf{KT}, gr(\tau) = gr(\tau')$, and $ag(\tau') \subseteq gr(\tau)$
3.d	$\tau \cup \tau' \in \mathsf{KT}$	if	$\tau, \tau' \in \mathsf{KT}$, and $gr(\tau) = gr(\tau')$

The various constraints on groups and modal agents play a role in the interpretation of action types, and will become clear in section 3.3. Also, they will be extensively discussed in section 3.7.

L_B is the 'learn' operator. $L_B\tau$ stands for 'group B learn that τ'. Instead of $L_{\{1,2,\dots,n\}}$ we write $L_{12\dots n}$. If $gr(\tau) = B$ then τ is called an *action type for group B* or a *B action type*. The operation 'local choice', that we mentioned in the introduction, is not an action type constructor but an action constructor. We therefore do not see it here. Observe that from the definitions it follows that $M, w \models [\tau]\varphi$ is undefined if $gr(M) \subset gr(\tau)$.

Definition 4 (FuncKT – **Model functional action types)**

$$\begin{array}{rcl} ?\varphi \in \mathsf{FuncKT} & \text{iff} & \varphi \in \mathsf{DKL} \\ L_B\tau \in \mathsf{FuncKT} & \text{iff} & \tau \in \mathsf{KT} \\ \tau \ ; \ \tau' \in \mathsf{FuncKT} & \text{iff} & \tau, \tau' \in \mathsf{FuncKT} \end{array}$$

Definition 5 (DetKT – **Deterministic action types)**

$$\begin{array}{rcl} ?\varphi \in \mathsf{DetKT} & \text{iff} & \varphi \in \mathsf{DKL} \\ L_B\tau \in \mathsf{DetKT} & \text{iff} & \tau \in \mathsf{DetKT} \\ \tau \ ; \ \tau' \in \mathsf{DetKT} & \text{iff} & \tau, \tau' \in \mathsf{DetKT} \end{array}$$

Observe that $\mathsf{DetKT} \subset \mathsf{FuncKT} \subset \mathsf{KT}$. Instead of $\tau \in \mathsf{FuncKT}$ we also say: τ is *model functional*, or just *functional*. Instead of $\tau \in \mathsf{DetKT}$ we also say: τ is *deterministic*. This terminology will become clear in the subsection on interpretation.

3.2 Examples

We give some examples of action types that can be interpreted in the model underlying the game state we described in the introduction: three players 1, 2 and 3 holding three cards red, white and blue, in that order. The public is $\{1, 2, 3\}$ and the set of atoms is $\{r_1, w_1, b_1, r_2, w_2, b_2, r_3, w_3, b_3\}$, for 'player 1 holds the red card', 'player 1 holds the white card', etc. The multiagent $S5$ model underlying this game state is hexa, see figure 1. String rwb names the

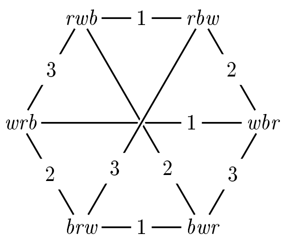

Figure 1: The model hexa, for three players each holding a card.

world where 1 holds red, 2 holds white and 3 holds blue, etc. As he holds red in both worlds, player 1 cannot distinguish world rwb from world rbw; hence the 1-link in the figure. We implicitly assume reflexivity and transitivity in figures of $S5$ models, such as hexa. Formula δ_{ijk} is the atomic description of world (dealing) ijk, e.g. $\delta_{rwb} := r_1 \land \neg w_1 \land \neg b_1 \land \neg r_2 \land w_2 \land \neg b_2 \land \neg r_3 \land \neg w_3 \land b_3$.

First, we give the example action types and their description in KT, then, we give some explanations on how they were constructed. The informal reading of the first two examples is ambiguous for player 3, as the group of those action types is smaller than the public $\{1, 2, 3\}$, the group of hexa.

1. player 1 holds the red card:
   $?r_1$
2. player 1 shows player 2 the red card:
   $L_{12}?r_1$
3. **example 1**: player 1 puts the red card on the table:
   $L_{123}?r_1$
4. **example 2**: player 1 shows (only) player 2 his card:
   $L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$
5. **example 3**: player 2 asks player 1 for the white card, 1 says he doesn't have it; 2 ends his move:
   $L_{123}?\neg w_1 ; L_{123}?\neg \mathsf{win}_2$
6. **example 4**: player 1 whispers in 2's ear a card that he (1) doesn't have:
   $L_{123}(L_{12}?\neg r_1 \cup L_{12}?\neg w_1 \cup L_{12}?\neg b_1)$

Ad 1: This test succeeds if atom r_1 holds. It is not specified what the players learn.

Ad 2: By showing his card to player 2, player 1 'reveals' that atom r_1 holds, in a way that both players have common knowledge of that fact: 2 observes 1 showing his card, 1 observes that 2 is observing it, etc. Thus the operator L_{12}. Still, one might consider two (equivalent) readings here: $L_{12}?K_1 r_1$ and $L_{12}?r_1$. We think that the second reading is more properly expressing what is happening here, and that the first reading is more appropriate for describing 'player 1 *tells* player 2 that he holds the red card'. The role of player 3 in this type of action is unclear.

Ad 3: Same as item 2, only now for group $\{1, 2, 3\}$, the public. Now, for every agent it is specified what he learns in this type of action, i.e. how his knowledge changes. This is the type of the action in example 1.

Ad 4: We paraphrase what happens here in greater precision: 'player 1 shows (only) player 2 his card' is the same as 'players 1, 2 and 3 learn that player 1 shows player 2 his card', which is the same as 'players 1, 2 and 3 learn that 1 shows the red card to 2, or the white card, or the blue card' which is the same as 'players 1, 2 and 3 learn that 1 and 2 learn that 1 holds red, or that 1 and 2 learn that 1 holds white, or that 1 and 2 learn that 1 holds blue'. This is expressed by the proposed KT type. Observe that this *still* doesn't describe the *action* of example 2, because in that case the red card was shown. That is an action, this is the action's type. Incidentally, in our analysis we have implicitly combined a question with an answer to it into an action type. We haven't modelled those separately, for that, see [vD99] or [vD00d].

Ad 5: The combination of the question and the answer is described by $L_{123}?\neg w_1$. As explained in example 3, ending one's move means making public that one doesn't know the dealing of cards. In hexa, there are six possible dealings of cards. Therefore, knowing one of them is described by $K_2\delta_{rwb} \vee K_2\delta_{rbw} \vee K_2\delta_{bwr} \vee K_2\delta_{brw} \vee K_2\delta_{wrb} \vee K_2\delta_{wbr}$. We abbreviate this formula as win_2. Therefore, 2 saying that he can't win is described by: $L_{123}?\neg\mathsf{win}_2$. Finally, we have two consecutive actions.

Ad 6: The analysis of example 4 is similar to that of example 2, only the three options are *not* having a card, instead of having a card. Again, this action type is not the same as the action in example 4, that specifies just one of the three actions of that type.

3.3 Local interpretation of action types

The, as we call it, 'public' interpretation of an A action type on an A model is defined in terms of the *local interpretation* on that A model of its B subprograms. The notion of local interpretation is the more basic one. The idea behind it is the following: 'public' means 'for all agents', 'local' means 'only for a certain group'.[2] To interpret a B action type τ in an A model 'locally', you 'forget' about the agents in $A \setminus B$. You may do that, because these other agents will be properly interpreted 'later', as action type τ can be a subprogram of an A action type, that can be publicly interpreted. In other words: we do not even *want* to interpret τ publicly: it just doesn't specify what the effects are for the agents outside it.

Note that it is fundamental, contrary to [Ger99], that when locally interpreting a B action type τ, knowledge changes for the agents not in B are *not*

[2]The term 'local' in 'local interpretation' might be somewhat prone to misunderstanding, but we hope that the reader appreciates our efforts to avoid calling it 'partial interpretation', as 'partial' seems a much overloaded term.

(yet) computed. The solution of [Ger99] is to enforce that agents not in B have learnt *nothing* from τ; we just say that they don't care *at this stage*.

The local interpretation of an action type τ on a knowledge model M consists of two relations:

- a relation $[\![\tau]\!]$ between knowledge models, i.e. pairs (M, M') where M' is a model resulting from τ's execution;
- a relation $[\![\tau]\!]$ between their worlds, to be more precise: pairs (w, w') that link every world $w' \in M'$ to a world $w \in M$, so that $[\![\tau]\!]^{-1}$ is a function from M' to M.

A relation between a world w in a model M and a world w' in a model M' should be seen as a relation between the knowledge states (M, w) and (M', w'). We have chosen to 'overload' the notation $[\![\cdot]\!]$ by using it both between models and between their worlds (between states).

As type of operation, local interpretation is not unlike bisimilarity. Bisimilarity is also both a relation $\underline{\leftrightarrow}$ between models and a relation $\mathfrak{R}$ between their worlds, although in the case of bisimilarity different symbols are used for the two different levels.

We now present the definition of local interpretation. Examples follow the definitions. We include an example, in section 3.5, that illustrates the differences with the approach of [Ger99]. We haven't encountered the notion of local interpretation of programs in the literature and regard it as our contribution to epistemic semantics.

Notations in the definition We use the following notations for infix binary relations R such as $[\![\cdot]\!]$ and $\rightarrow_\tau$: $[aR] := \{b \mid aRb\}$, as in clause 6.a.2, and $aRbRc := (aRb \text{ and } bRc)$, as in clause 6.$c$.3. In the clauses under (e), write $M'' = \langle W'', (\sim''_a)_{a \in gr(\tau)}, V'' \rangle$ for an arbitrary $M'' \in [M[\![\tau]\!]]$; in 6.e.2 and 6.e.4 we thus refer to access and valuation in M''. In 6.e.3 we use the following shorthand, given a context of two models: $[\![\tau]\!]^{-1}(w) := \iota v.(M, v)[\![\tau]\!](M'', w)$. I.e. $[\![\tau]\!]^{-1}(w)$ is the unique world in M that is the origin of world w in M''. (See proposition 1, on page 22.)

Definition 6 (Local interpretation of an action type)
The local interpretation $[\![\cdot]\!]$ of a KT action type is a relation between knowledge models and between their worlds, defined by simultaneous induction. Let $M = \langle W, (\sim_a)_{a \in A}, V \rangle$ be a knowledge model, where $A \in \mathbf{A}$. Assume that both the agents and the group of the action type to be interpreted are a subset of or equal to A. Then:

$$
\begin{array}{llll}
6.a.1 & M\llbracket ?\varphi \rrbracket M' & \text{iff} & M' = \langle W', \emptyset, V{\restriction}W' \rangle \\
 & & & \quad \text{where } W' = \{ w \in W \mid M, w \models \varphi \} \\
6.a.2 & M\llbracket L_B \tau \rrbracket M' & \text{iff} & M' = \bigoplus_B [M\llbracket \tau \rrbracket] \text{ as defined below} \\
6.a.3 & M\llbracket \tau \ ; \ \tau' \rrbracket M' & \text{iff} & \exists M'' : M\llbracket \tau \rrbracket M'' \llbracket \tau' \rrbracket M' \\
6.a.4 & M\llbracket \tau \cup \tau' \rrbracket M' & \text{iff} & M\llbracket \tau \rrbracket M' \text{ or } M\llbracket \tau' \rrbracket M' \\
 & & & \\
6.b.1 & (M, w)\llbracket \tau \rrbracket (M', w') & \text{iff} & M\llbracket \tau \rrbracket M' \text{ and } w \rightarrow_\tau w' \\
 & & & \\
6.c & & & \forall M, M' : M\llbracket \tau \rrbracket M', \forall w \in M, \forall w' \in M' : \\
 & & & \\
6.c.1 & w \rightarrow_{?\varphi} w' & \text{iff} & w = w' \\
6.c.2 & w \rightarrow_{L_B \tau} w' & \text{iff} & \exists M'' \in [M\llbracket \tau \rrbracket] : \exists w'' \in M'' : \\
 & & & w' = (M'', w'') \text{ and } w \rightarrow_\tau w'' \\
6.c.3 & w \rightarrow_{\tau ; \tau'} w' & \text{iff} & \exists M'' : M\llbracket \tau \rrbracket M'' \llbracket \tau' \rrbracket M', \exists w'' \in M'' : \\
 & & & w \rightarrow_\tau w'' \rightarrow_{\tau'} w' \\
6.c.4 & w \rightarrow_{\tau \cup \tau'} w' & \text{iff} & w \rightarrow_\tau w' \text{ or } w \rightarrow_{\tau'} w' \\
 & & & \\
6.d.1 & \bigoplus_B [M\llbracket \tau \rrbracket] & = & \langle W', (\sim'_a)_{a \in B}, V' \rangle, \text{ where:} \\
 & & & \\
6.e.1 & W' & = & \{ (M'', w) \mid w \in M'' \in [M\llbracket \tau \rrbracket] \} \\
6.e.2 & \forall a \in gr(\tau) : & & \forall (M'', w), (M'', w') \in W' : \\
 & & & (M'', w) \sim'_a (M'', w') \quad \text{iff} \quad w \sim''_a w' \\
6.e.3 & \forall a \in B \setminus gr(\tau) : & & \forall (M'', w), (M^*, w') \in W' : \\
 & & & (M'', w) \sim'_a (M^*, w') \quad \text{iff} \\
 & & & \llbracket \tau \rrbracket^{-1}(w) \sim_a \llbracket \tau \rrbracket^{-1}(w') \\
6.e.4 & \forall (M'', w) \in W' : & & V'_{(M'', w)} = V''_w
\end{array}
$$

Ad (a) In part (a) of definition 6, the interpretation of action types on models is defined.

Ad (b) and (c) In parts (b) and (c) of definition 6, the interpretation of action types on states is defined. Clause $6.b.1$ states that to interpret an action type τ in a state (M, w), one has to interpret τ in the model M *and* one has to determine the images of w under that interpretation. The operator $\rightarrow_\tau$ determines what these images are. Clause $6.c.2$ states that world w' is an image of world w for type $L_B\tau$, if w' is a state (M'', w'') that results from interpreting τ on (M, w).

Ad (d) and (e) In parts (d) and (e) of definition 6, we define the model $\bigoplus_B[M\llbracket \tau \rrbracket]$, the result of interpreting an action of type $L_B\tau$. The model $\bigoplus_B[M\llbracket \tau \rrbracket]$ is the direct sum $\bigoplus[M\llbracket \tau \rrbracket]$, plus access added for the agents $b \in B \setminus gr(\tau)$, according to clause $6.e.3$ of the definition. (So that, indeed, B is just the parameter we need.) To construct the direct sum $\bigoplus \mathcal{M}$ of a set $\mathcal{M}$ of models, we cannot just take the union of their domains. We have to pair a world $w \in M_i \in \mathcal{M}$ to some index i in order to determine the model M_i it originates from. The pair (w, i) is then a world in the sum model. We have chosen to take the model *itself* as index. Thus we get: (w, M_i). As we might as well write (M_i, w), this amounts to taking *states* for models from $\mathcal{M}$ as *worlds* in the direct sum $\bigoplus \mathcal{M}$.

Unless confusion results, in the examples we *still* simply take the union of the domains of models in $\bigoplus_B [M[\![\tau]\!]]$. In the examples we name worlds by card dealings, that correspond to the atomic descriptions of those worlds. Even when different worlds have the same name, we can distinguish them by the different access they have to other worlds. [3]

It is *essential* that the local interpretation of an action type τ in a model M is only defined if both $ag(\tau) \subseteq gr(M)$ and $gr(\tau) \subseteq gr(M)$. Without the first, a test formula with modal operators not in $gr(\tau)$ cannot be evaluated in M (see clause 6.a.1 of the definition). Without the second, there are agents in a group B learning something in τ, that do not occur in M, so that their access in the resulting model cannot be computed (see clause 6.e.3 of the definition). See also section 3.7.

Definition 7 (Executable)
An action type τ is *executable* in a knowledge model M, if the local interpretation of τ on M is defined and is not the empty relation. A action type τ is *executable* in a knowledge state (M, w), if the local interpretation of τ in (M, w) is defined and is not the empty relation: i.e. if it is executable in M and if w has a τ-image under that interpretation.

Definition 8 (Equivalence)
Two KT action types τ and τ' are *equivalent*, notation $\tau = \tau'$, if they have the same local interpretation on all models: i.e. if $[\![\tau]\!] = [\![\tau']\!]$.

Definition 9 (Public interpretation)
If $gr(\tau) = gr(M)$ then $[\![\tau]\!]$ is the public interpretation of τ in M.

These three definitions will later be extended to include actions.

Notation If the local interpretation of an action type τ is a functional relation on the level of knowledge models, we write $M[\![\tau]\!]$ for the unique M' such that $M[\![\tau]\!]M'$. The model functional action types FuncKT have such an interpretation. (See subsection 3.8.)

[3]By indexing worlds with models, we *would* still run into naming trouble when pairing a world with two models with the same name. For technical reasons, different models will never get the same name. This will only become clear in section 4. Still, when computing $[M[\![\tau]\!]]$ the same model might also be multiply 'produced by τ', e.g. when $\tau = \tau' \cup \tau'$. We have taken an easy way out of that problem: we are only interested in the *set* $[M[\![\tau]\!]]$, so that multiple occurrences of models have been deleted. There is also a 'difficult' way out of that problem: a model constructed from the set of renamed multiple occurences would be bisimilar to one constructed from the set with multiple occurrences deleted, see section 3.8 for some results concerning bisimilarity.

If, beyond that, the relation between worlds of knowledge models of the interpretation of an action type τ is functional, we write $(M, w)[\![\tau]\!]$ for the unique (M', w') such that $(M, w)[\![\tau]\!](M', w')$. There is such a unique (M', w') if τ is not only model functional, but also deterministic, i.e. if $\tau \in \mathsf{DetKT}$. (See subsection 3.8.) If τ is deterministic, instead of $(M, w)[\![\tau]\!]$ we might as well write $(M[\![\tau]\!], w[\![\tau]\!])$. This notational convention will also be used for knowledge actions, to be defined in the next section.

Instead of $\bigoplus_B\{M_1, M_2, ..., M_n\}$ we also write $M_1 \bigoplus_B M_2 \bigoplus_B ... \bigoplus_B M_n$.

Instead of a 'two-level' definition between both models and their worlds (seen as states), we considered defining local interpretation as a relation between knowledge states *only*, which would in this respect have been as in [BMS98], [Bal99] and [Ger99]. However, the basic programming operations 'test' and 'learning' can be much more elegantly defined between models. For the relation between interpretation on models and on worlds, see also subsection 3.8 on action type properties.

3.4 Examples

We now illustrate definition 6 by computing the interpretation of the example action types from subsection 3.2 on hexa, the knowledge model of figure 1, for three players holding three cards red, white and blue. All examples can be publicly interpreted.

Example 1
Player 1 puts the red card on the table: $L_{123}?r_1$.

We apply definition 6 stepwise:

$\mathsf{hexa}[\![L_{123}?r_1]\!]M$
$\Leftrightarrow$ definition 6.a.2
$M = \bigoplus_{123}[\mathsf{hexa}[\![?r_1]\!]]$

$\mathsf{hexa}[\![?r_1]\!]M'$
$\Leftrightarrow$ definition 6.a.1
$M' = \langle\{rwb, rbw\}, \emptyset, V\restriction\{rwb, rbw\}\rangle$

As M' is unique, we may write $M' = \mathsf{hexa}[\![?r_1]\!]$ and we have that $[\mathsf{hexa}[\![?r_1]\!]] = \{\mathsf{hexa}[\![?r_1]\!]\}$. We still have to compute the relations between worlds in hexa and in $\mathsf{hexa}[\![?r_1]\!]$. According to clause 6.$c$.1: $rwb \rightarrow_{?r_1} rwb$ and $rbw \rightarrow_{?r_1} rbw$. We can now compute M ($= \mathsf{hexa}[\![L_{123}?r_1]\!]$). Write $M = \langle W^M, \{\sim_1^M, \sim_2^M, \sim_3^M\}, V^M\rangle$. According to clause 6.$e$.1: $W^M = \{(\mathsf{hexa}[\![?r_1]\!], rwb), (\mathsf{hexa}[\![?r_1]\!], rbw)\}$. As $gr(?r_1) = \emptyset$, clause 6.e.2 doesn't apply. For the other agents, i.e. for all agents 1,2 and 3, we continue as follows:

$(\mathsf{hexa}[\![?r_1]\!], rwb) \sim_1^M (\mathsf{hexa}[\![?r_1]\!], rbw)$
$\Leftrightarrow$ definition 6.e.3

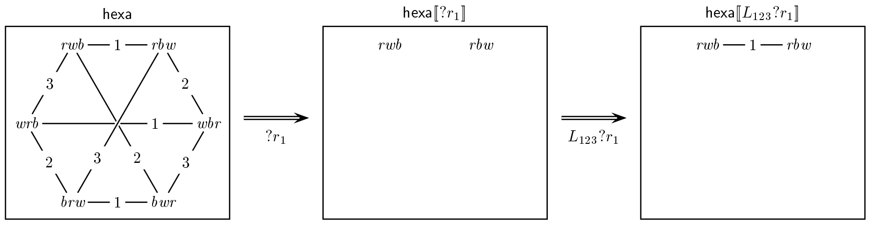

Figure 2: Computing $\mathsf{hexa}\llbracket L_{123}?r_1 \rrbracket$

$\llbracket ?r_1 \rrbracket^{-1}(rwb) \sim_1 \llbracket ?r_1 \rrbracket^{-1}(rbw)$
$\Leftrightarrow$ definition 6.b.1 and 6.c.1
$rwb \sim_1 rbw$

If we simplify the notation of domain W^M of model $\mathsf{hexa}\llbracket L_{123}?r_1 \rrbracket$, as suggested in the remarks ad clause 6.e.1, we even get:

$$rwb \sim_1^M rbw \quad \text{iff} \quad rwb \sim_1 rbw$$

The six reflexive links in the model $\mathsf{hexa}\llbracket L_{123}?r_1 \rrbracket$ (for 1, 2, and 3, for both rwb and rbw) are similarly computed. Further, clause 6.e.4 states that the valuation V^M on the 'new' worlds rwb and rbw is the same as on the 'old' ones. Figure 2 gives an overview of our computations.

Example 2
Player 1 shows his card to player 2: $L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$.

We elaborate on the necessary computations:

$\mathsf{hexa}\llbracket L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1) \rrbracket M'$
$\Leftrightarrow$ definition 6.a.2
$M' = \bigoplus_{123}[\mathsf{hexa}\llbracket L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1 \rrbracket]$

$\mathsf{hexa}\llbracket L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1 \rrbracket M''$
$\Leftrightarrow$ definition 6.a.4
$\mathsf{hexa}\llbracket L_{12}?r_1 \rrbracket M''$ or $\mathsf{hexa}\llbracket L_{12}?w_1 \rrbracket M''$ or $\mathsf{hexa}\llbracket L_{12}?b_1 \rrbracket M''$

We assume that $\cup$ is associative, which is proven later, in proposition 2. We get three different models M'' this way. We compute the first one.

$\mathsf{hexa}\llbracket L_{12}?r_1 \rrbracket M''$
$\Leftrightarrow$ definition 6.a.2
$M'' = \bigoplus_{12}[\mathsf{hexa}\llbracket ?r_1 \rrbracket]$

Similarly to the computation in the previous example, we get $M'' = \mathsf{hexa}[\![L_{12}?r_1]\!] = \langle\{rwb, rbw\}, (\sim_1^{M''}, \sim_2^{M''}), V^{M''}\restriction\{rwb, rbw\}\rangle$. More strictly, instead of $\{rwb, rbw\}$, the domain is $\{(\mathsf{hexa}[\![?r_1]\!], rwb), (\mathsf{hexa}[\![?r_1]\!], rbw)\}$. Apart from reflexive access for both 1 and 2 in both worlds, we only have $rwb \sim_1 rbw$. Access for player 3 is not computed. Similarly we compute $\mathsf{hexa}[\![L_{12}?w_1]\!]$ and $\mathsf{hexa}[\![L_{12}?b_1]\!]$, see figure 3.

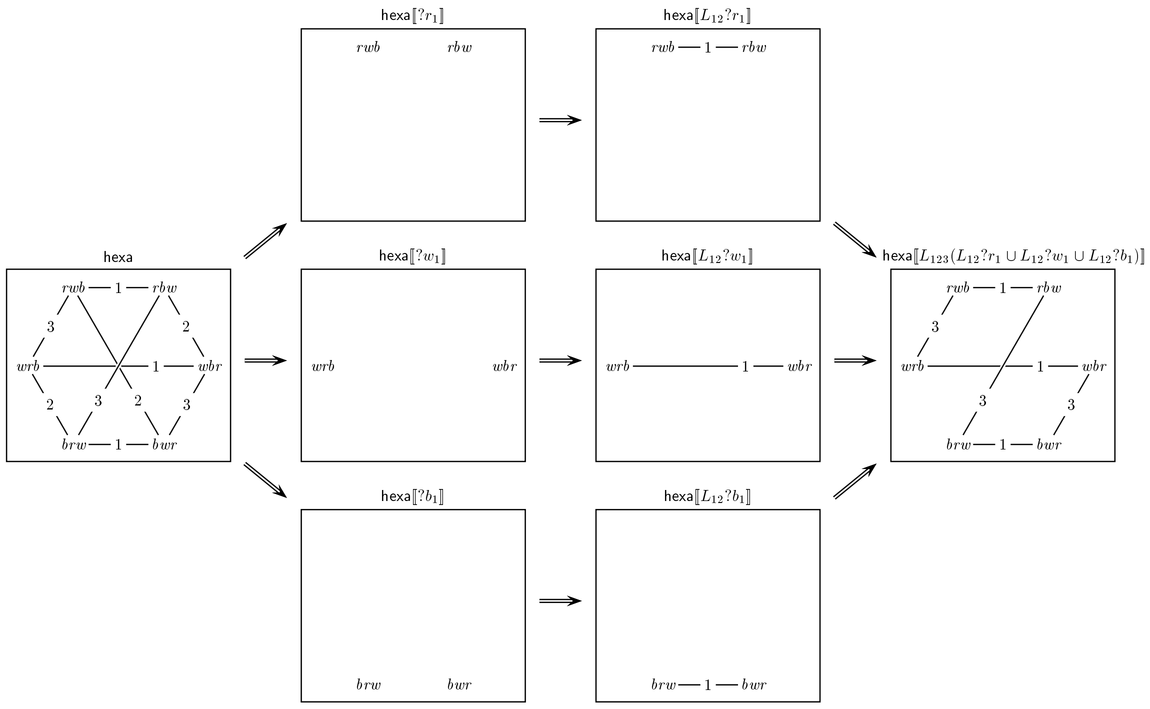

Figure 3: Computing $\mathsf{hexa}[\![L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)]\!]$

We finish the computation by determining

$$\begin{aligned} M' &= \mathsf{hexa}[\![L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)]\!] \\ &= \mathsf{hexa}[\![L_{12}?r_1]\!] \bigoplus_{123} \mathsf{hexa}[\![L_{12}?w_1]\!] \bigoplus_{123} \mathsf{hexa}[\![L_{12}?b_1]\!] \end{aligned}$$

Formally, the domain W' of M' is the set

$$\begin{aligned} W' = \{ &(\mathsf{hexa}[\![L_{12}?r_1]\!], (\mathsf{hexa}[\![?r_1]\!], rwb)), \\ &(\mathsf{hexa}[\![L_{12}?r_1]\!], (\mathsf{hexa}[\![?r_1]\!], rbw)), \\ &(\mathsf{hexa}[\![L_{12}?w_1]\!], (\mathsf{hexa}[\![?w_1]\!], wrb)), \\ &(\mathsf{hexa}[\![L_{12}?w_1]\!], (\mathsf{hexa}[\![?w_1]\!], wbr)), \\ &(\mathsf{hexa}[\![L_{12}?b_1]\!], (\mathsf{hexa}[\![?b_1]\!], bwr)), \\ &(\mathsf{hexa}[\![L_{12}?b_1]\!], (\mathsf{hexa}[\![?b_1]\!], brw)) \} \end{aligned}$$

Again, for convenience we simplify it to $W' = \{rwb, rbw, wrb, wbr, bwr, brw\}$.

Access for 1 and 2 remains as it is, using definition 6.e.2. Similarly to the computations for access in example 1, we have reflexive access for player 3, and apart from that we can compute in M' that $rwb \sim_3 wrb$, $rbw \sim_3 brw$, and $bwr \sim_3 wbr$. Figure 3 visualizes the resulting model, and how we have constructed it.

In any world of the resulting model, player 2 knows the dealing of cards. Player 1 doesn't know the cards of 2 and 3, although he knows that 2 knows it.

Example 3
Everybody learns that player 1 doesn't have the white card, and everybody learns that player 2 can't win the game (doesn't know the dealing of cards) after that: $L_{123}?\neg w_1 \ ; \ L_{123}?\neg \mathsf{win}_2$.

We elaborate on the necessary computations. First we interpret $L_{123}?\neg w_1$ on hexa. The test $?\neg w_1$ succeeds on four of the six worlds of hexa. As before, access for all three players is then reestablished between worlds that were already linked. Apart from reflexive links for all players between all worlds, we get that $rwb \sim_2 bwr$ and that $brw \sim_3 rbw$.

We continue by interpreting $L_{123}?\neg \mathsf{win}_2$ on $\mathsf{hexa}[\![L_{123}?\neg w_1]\!]$ (as we have that $M[\![\tau \ ; \ \tau']\!]M' \Leftrightarrow M[\![\tau \ ]\!][\![\tau']\!]M'$). The formula win_2 describes that player 2 knows which of the six possible dealings of cards is actually the case, in game terms: that 2 can win. In semantical terms this condition is only satisfied in singleton equivalence classes of 2's access. As we test on 2 not being able to win, we therefore remove all singletons in 2's access. Next we reassert previously (in $\mathsf{hexa}[\![L_{123}?\neg w_1]\!]$) existing access for 1, 2 and 3 between the remaining worlds rwb and bwr. This is general procedure for tests that are publicly learnt, i.e. 'public announcements' in terms of [BMS98]. Figure 4 pictures the resulting model, and how we have constructed it.

In the final model $\mathsf{hexa}[\![L_{123}?\neg w_1 \ ; \ L_{123}?\neg \mathsf{win}_2]\!]$ on the right in figure 4, both 1 and 3 have full knowledge of the dealing of cards, whereas in any state of $\mathsf{hexa}[\![L_{123}?\neg w_1]\!]$ consistent with that, only 3 can win. So 1 can win because 2 says he (2) can't.

Example 4
Player 1 whispers in 2's ear a card that he (1) doesn't have: $L_{123}(L_{12}?\neg r_1 \cup L_{12}?\neg w_1 \cup L_{12}?\neg b_1)$.

Unlike the previous examples, this is a case of true nondeterminism: 1 can choose what to whisper. Now the model resulting from the execution of an action type can be more complex, in terms of the its number of worlds, than the model upon which the type is executed. Indeed, this is here the case. We do not perform the computations in detail.

Figure 5 pictures $\mathsf{hexa}[\![L_{123}(L_{12}?\neg r_1 \cup L_{12}?\neg w_1 \cup L_{12}?\neg b_1)]\!]$. Note that we assume the accessibility relations to be transitive. As in the previous examples, for improved readability we haven't renamed the worlds according to clause

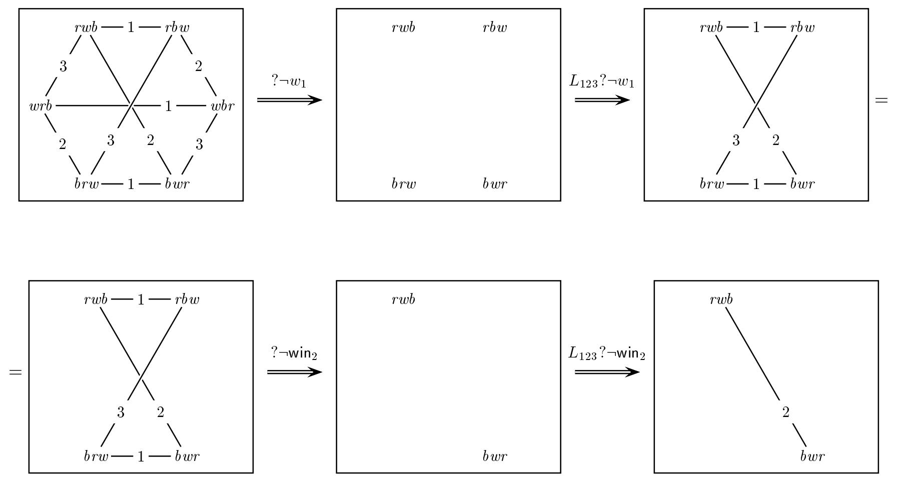
Figure 4: Computing $\mathsf{hexa}[\![L_{123}?\neg w_1 \ ; \ L_{123}?\neg \mathsf{win}_2]\!]$

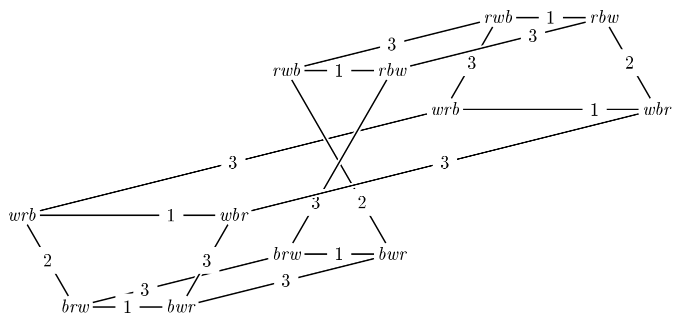
Figure 5: Picture of $\mathsf{hexa}[\![L_{123}(L_{12}?\neg r_1 \cup L_{12}?\neg w_1 \cup L_{12}?\neg b_1)]\!]$

6.e.1, but have kept their original names. Otherwise, e.g. the world wrb 'in front' would be named $(\mathsf{hexa}[\![L_{12}?\neg r_1]\!], (\mathsf{hexa}[\![?\neg r_1]\!], wrb))$, call it wrb_{front}, and the world wrb 'at the back', would be named $(\mathsf{hexa}[\![L_{12}?\neg b_1]\!], (\mathsf{hexa}[\![?\neg b_1]\!], wrb))$, call it wrb_{back}. World wrb in hexa has both wrb_{front} and wrb_{back} as images under the interpretation of this type, corresponding to the choice he can make between red and blue. This can be checked by computing both $wrb \rightarrow_\tau wrb_{\mathsf{front}}$ and $wrb \rightarrow_t auwrb_{\mathsf{back}}$, for type $\tau = L_{123}(L_{12}?\neg r_1 \cup L_{12}?\neg w_1 \cup L_{12}?\neg b_1)$, by stepwise application of clauses from definition 6.c.

Without renaming, we can still distinguish the world wrb in front from the world wrb at the back, because player 2 is less informed in front. In other words: because the worlds differ in their access to other worlds.

For more examples of types of action in hexa, such as players suspecting other players of cheating, and for examples of action types in related models for card games, see [vD00c].

In section 2.3 we mentioned that formulas $[?K_1p]K_1p$ are undefined on any model. Why, will now be clear: note that $\mathsf{hexa}, rwb \models [?K_1r_1]K_1r_1 \Leftrightarrow \mathsf{hexa}[\![?K_1r_1]\!], rwb \models K_1r_1$. The model $\mathsf{hexa}[\![?K_1r_1]\!]$ is the same as $\mathsf{hexa}[\![?r_1]\!]$, as in figure 2. As it is an $\emptyset$ model, obviously we cannot interpret the formula K_1r_1.

We do not see this as a limitation of our approach. Instead, we see it as an advantage that we that, e.g. show that $\mathsf{hexa} \models [?K_1r_1]r_1$. Put in other words: some formulas can be locally interpreted without any assumptions on other agents. This is different from the approach in [Ger99], where one has to assume that all other agents learn nothing. We continue with an example of that.

3.5 Local interpretation versus updating

In example 2 we computed the (public) interpretation of $L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$ in hexa. Instead of (locally) interpreting subprogram $L_{12}?r_1$, as in figure 3, we might have considered 'totally' interpreting that subprogram, before continuing to incorporate its interpretation in the interpretation of $L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$. How to proceed? Because we do not know how the execution of this type of action effects player 3, we might as well assume that he (his knowledge) is *not at all* affected by it. This is the approach in [Ger99]. In [Ger99], DEL programs are always totally interpreted in this manner. Also, for technical reasons, they can only be interpreted in *states*, not in models. The action type $L_{12}?r_1$ corresponds to the DEL program $?r_1 \ ; \ U_{12}?r_1$. This expresses that: 1 and 2 have learnt that 1 holds the red card, and 3 hasn't learnt anything. Figure 6 pictures the state resulting from interpreting $?r_1 \ ; \ U_{12}?r_1$ in state (hexa, rwb). Continuing in this manner, a DEL program corresponding to $L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$ has the same interpretation as that action type in every state of hexa.

For the purpose of computing relations between $S5$ models, we think the approach of [Ger99] less attractive, for the following three reasons:

- Intermediate in the computation are models that are not $S5$, such as the model in figure 6.
- Intermediate in the computation are models that are more complex than the initial and the final model. The model in figure 6 contains 8 worlds, corresponding to 8 non-bisimilar states. Both hexa and the resulting model $\mathsf{hexa}[\![L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)]\!]$ consist of 6 worlds.
- One has to assume an actual state (the point of the model), before one can compute the effect of a program (an action). All six states of hexa

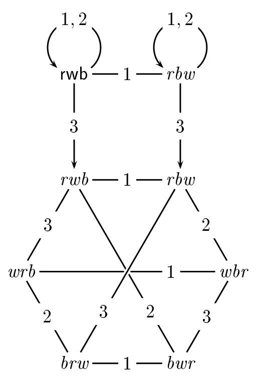

Figure 6: Picture of (hexa, rwb)$[\![?r_1 \ ; \ U_{12}?r_1]\!]$, using [Ger99]. The designated world rwb is in roman style. This is *not* an example of local interpretation.

are compatible with ('survive') the execution of $L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$. That means six (although similar) computations.

But, of course, a more essential difference is that one can locally interpret programs (actions or types) without any assumptions on the other agents, i.e. without assuming that the other agents have learnt nothing. See also [vD00b].

3.6 Another example

There is only one atom p and there are only two agents 1 and 2. Letter is the model $\langle\{w, w'\}, (\sim_1, \sim_2), V\rangle$ with both $\sim_1$ and $\sim_2$ the universal relation, and with $V_w(p) = 1, V_{w'}(p) = 0$. We can think of w as the world where 1 gets 'an invitation for a night out in Amsterdam' and of w' as the world where, instead, 1 is invited, or, if he wishes to see it that way, 'obliged to give a lecture'. It is commonly known that these are the only alternatives. We can also think of this example as the knowledge game for two players and two cards, where the players haven't yet looked into their cards. Three examples of action types in this model are:

- An outsider tells the agents that agent 1 is invited for a night out in Amsterdam[4]:
  $L_{12}?p$
- In a letter is written that agent 1 is invited for a night out in Amsterdam. Agent 1 is given the letter, opens it, and reads its contents. Agent 2 observes player 1 opening his letter (and this is commonly known):
  $L_{12}(L_1?p \cup L_1?\neg p)$

[4]Which results in the same model as: In a letter is written that agent 1 is invited for a night out in Amsterdam. Agent 1 is given the letter, opens it, and reads its contents aloud.

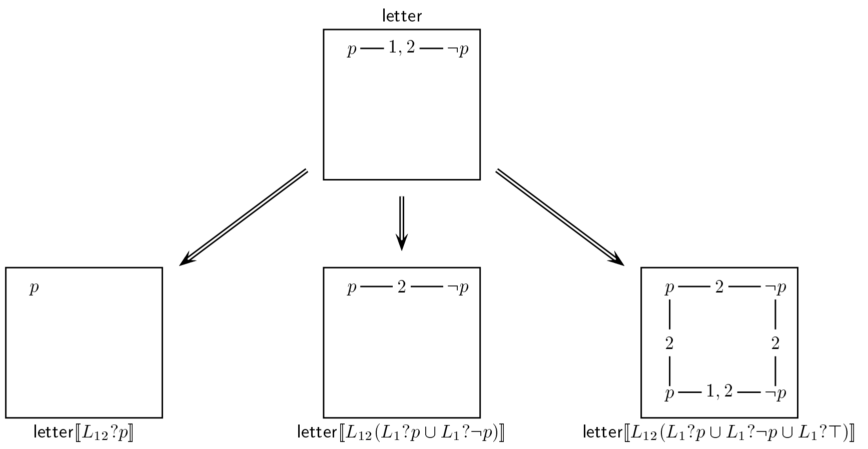

Figure 7: Nightclub or lecture

- Same as previous item, only now agent 2 doesn't know but only suspects 1 to have opened the letter (and this is commonly known): $L_{12}(L_1?p \cup L_1?\neg p \cup L_1?\top)$

Figure 7 presents the models produced by these action types. Instead of w and w', we named the worlds by their atomic descriptions p and $\neg p$.

3.7 Motivations for the constraints on KT

Given the interpretation of action types, and the examples, we can now explain the constraints on action type construction in definition 3. Without these constraints, local interpretation can be incorrect, from the viewpoint of expected postconditions, or undefined.

Definition 3.a: tests

There are no constraints on tests. For those who oppose to the name 'test': see the end of this subsection.

Definition 3.b: learning

In clause 3.b we required that $gr(\tau) \subset B$. We can relax this constraint to $gr(\tau) \subseteq B$. Clause 6.c.2, on the interpretation of the learning operator, in definition 6, will still construct a correct model. Relaxing the constraint doesn't have any (modelling) value, however. If $gr(\tau) = B$, the interpretation of learning is either identity or constructs a direct sum. Examples:

$[\![L_B(L_B\tau)]\!]$ is the identity on all models where it can be executed. There is nothing to be gained from learning that you learn.

Because $\bigoplus_\emptyset$ is simply $\bigoplus$, the model $\mathsf{hexa}[\![L_{123}(L_{123}?r_1 \cup L_{123}?w_1)]\!]$ would be the direct sum $\mathsf{hexa}[\![L_{123}?r_1]\!] \bigoplus \mathsf{hexa}[\![L_{123}?w_1]\!]$. From the 'viewpoint' of its agents the difference doesn't matter. For all worlds in $\mathsf{hexa}[\![L_{123}?r_1]\!]$, state $(\mathsf{hexa}[\![L_{123}?r_1]\!], w)$ is modally equivalent to state $(\mathsf{hexa}[\![L_{123}(L_{123}?r_1 \cup L_{123}?w_1)]\!], w)$, because the latter consists of two disconnected parts.

Similar to the case $B = gr(\tau)$, also for $B \subset gr(\tau)$ is the interpretation of type $L_B\tau$ the same as that of τ.

Definition 3.c: sequential execution

Given an action type $\tau \ ; \ \tau'$, we required both that $gr(\tau) = gr(\tau')$ and that $ag(\tau') \subseteq gr(\tau)$. The result of interpreting τ will be some model M with access defined for the agents in $gr(\tau)$.

If $B = gr(\tau) \subset gr(\tau') = C$, $[\![\tau']\!]$ would be *undefined.* Action type τ' would contain a subprogram of type $L_C\tau^*$ that cannot be interpreted, because access for agents $a \in C \setminus B$ can not be defined from a-access between worlds from M, as access for a is undefined on M. An example: The interpretation of the action type $L_2?r_1 \ ; \ L_{123}?r_1$ is undefined, because access for 1 and 3 cannot be computed.

If, on the other hand, $C = gr(\tau') \subset gr(\tau) = B$, then the result of interpreting $\tau \ ; \ \tau'$ in a $B \cup C$ model would *incorrectly* be called the public interpretation of that action type in that model, even though $gr(\tau \ ; \ \tau') = gr(M)$. Any model resulting from executing $\tau \ ; \ \tau'$ would only be a C model, so that we can't compute the knowledge of agents not in C, as a result of its execution. An example: the interpretation of the action type $L_{123}?r_1 \ ; \ L_2?r_1$ is defined but incorrect. E.g. in the resulting model M', e.g. $M' \models K_3r_1$ is undefined, but should actually hold. Note that, on the other hand, model $M[\![L_{123}?r_1]\!][\![L_2?r_1]\!]$ is defined. (Only if $gr(\tau) = gr(\tau')$ it holds that $[\![\tau]\!][\![\tau']\!] = [\![\tau \ ; \ \tau']\!]$.)

If τ contains modal agents not in the group of τ, i.e. if $D = (ag(\tau') \setminus gr(\tau)) \neq \emptyset$, then τ' contains a test on a proposition with a modal operator for agents that cannot be evaluated on M. This modal operator must be either a modal operator K_a, with $a \in D$, or a modal operator C_B, with $B \cap D \neq \emptyset$, or a modal operator $[\tau^*]$ with modal agents in D. A $gr(\tau)$ model M on which such propositions are to be evalutated, lacks access for those agents. An example: the interpretation of the action type $L_1?K_2r_1 \ ; \ L_1?K_2r_1$ is undefined, because the test on K_2r_1 in the second occurrence of $L_1?K_2r_1$ cannot be evaluated in the 1-model that results from interpreting the first occurrence of that formula.

For those who feel uncomfortable under this restriction: the following explanation may help to make it seem reasonable. In the example above the first occurrence of $L_1?K_2r_1$ is interpreted locally because we expect it to be a subprogram of an action type for a larger group that contains 2, so that we can compute what 2 has learnt from this subprogram. Until we have specified that action type, we *want* 2's access to be undefined, and therefore wouldn't by any means *want* to refer to 2's knowledge, which is exactly what happens in the second occurrence of $L_1?K_2r_1$. So basically the constraint $ag(\tau') \subseteq gr(\tau)$ rules out 'bad programming'.

As a matter of fact, we considered deleting the ';' operation from our language. When specifying players' communication in games, any sequential execution can be described on the level of the public: $\tau \; ; \; \tau'$ is restricted to sequences of **A** action types *only.* This would have simplified our definitions considerably.

Definition 3.d: nondeterministic choice

For $\tau \cup \tau'$ to be an action type, we required that $gr(\tau) = gr(\tau')$. If not, constructs of type $L_B(\tau \cup \tau')$ might be *incorrectly* interpreted for agents $a \notin gr(\tau) \cap gr(\tau')$. An example: $L_{123}(L_2?r_1 \cup L_3?w_1)$ would be incorrectly interpreted in hexa. As $gr(L_2?r_1 \cup L_3?w_1) = \{2, 3\}$, when constructing $\mathsf{hexa}[\![L_{123}(L_2?r_1 \cup L_3?w_1)]\!]$ from $\mathsf{hexa}[\![L_2?r_1 \cup L_3?w_1]\!]$ only access for 1 is added. In particular, no 3-access is added to $\mathsf{hexa}[\![L_2?r_1]\!]$ and no 2-access is added to $\mathsf{hexa}[\![L_3?w_1]\!]$. E.g., if the actual state was rwb, in the model resulting from the execution of this action player 3 cannot imagine rwb to be the actual dealing of cards. The model is therefore not even $S5$.

This model is obviously not the intended interpretation: 3 learns that either 2 learns r_1 or that he himself learns w_1. Therefore, if after execution of this action he has not learnt w_1, which can be checked by introspection, he knows that 2 must have learnt r_1 instead. As 3 also knows his own card, he now knows all three cards: 3 cannot just imagine rwb, 3 even knows this dealing to be the case.

Why is a test a test?

Why is a KT test called a test? Tests supposedly do not change the objects they operate upon, but just let them survive, or not. A test $?\varphi \in \mathsf{KT}$ is only a test in this sense, if it can be publicly interpreted, *not* if it is only locally interpreted. Tests can only be publicly interpreted on models without access. In that case the test formulas must not contain modal operators. Even then, they only 'do not change the model' if that model consists of a singleton world. An example of a test that 'doesn't change the model': $?r_1$ can be publicly interpreted on the model $\langle \{rwb\}, \emptyset, V \upharpoonright \{rwb\} \rangle$, with V and rwb as in hexa. A counterexample, i.e. a 'test that changes the model': the test $?K_1 r_1$ can never be *publicly* interpreted, because $\emptyset = gr(?K_1 r_1) \subset ag(?K_1 r_1) = \{1\}$: it can only be publicly interpreted on an $\emptyset$ model, but on such a model we cannot interpret modal operator K_1. It can be locally interpreted on hexa, in which case it *does* change the model it operates upon. The resulting model $\mathsf{hexa}[\![?K_1 r_1]\!]$ is identical to $\mathsf{hexa}[\![?r_1]\!]$, as computed in figure 3.

One might remark that the word 'test' is therefore inappropriate for this KT construct. We agree. (But do not have a better name.)

3.8 Action type properties

In this section we prove some elementary and useful properties of action types.

Proposition 1 (Local interpretation on models versus states)
Let M be a knowledge model. Let (M, w) be a knowledge state. Let $\tau \in \mathsf{KT}$. Then:

$$(a) \quad \forall M' : M[\![\tau]\!]M' \Rightarrow \forall w' \in M' : \exists! w^* \in M : (M, w^*)[\![\tau]\!](M', w')$$

$$(b) \quad \forall M' : \forall w' : (M, w)[\![\tau]\!](M', w') \Rightarrow M[\![\tau]\!]M'$$

Proof

(a) Induction on τ. E.g. case $L_B\tau$: Suppose $M[\![L_B\tau]\!]M'$. Let $w' \in M'$. As $M' = \bigoplus_B[M[\![\tau]\!]]$, it holds that $w' = (M'', w'')$ for some $M'' \in [M[\![\tau]\!]]$ and $w'' \in M''$. By induction there must be exactly one $w \in M$ such that $(M, w)[\![\tau]\!](M'', w'')$. From definition $6.b.1$ follows $(M, w) \rightarrow_\tau (M'', w'')$. From that and from definition $6.c.2$ follows $(M, w) \rightarrow_{L_B\tau} (M', w')$, i.e. $(M, w) \rightarrow_{L_B\tau} (M', (M'', w''))$. ■

(b) Immediately from definition 6.b.1. ■

Because of proposition 1, in clause $e.3$ of definition 6 we can write $[\![\tau]\!]^{-1}(w')$ for the unique origin of a world w'.

Proposition 2 (Associativity of action type construction operations)

$$(a) \quad \forall \tau, \tau', \tau^* \in \mathsf{KT} : (\tau \cup \tau') \cup \tau^* = \tau \cup (\tau' \cup \tau^*)$$

$$(b) \quad \forall \tau, \tau', \tau^* \in \mathsf{KT} : (\tau \ ; \ \tau') \ ; \ \tau^* = \tau \ ; \ (\tau' \ ; \ \tau^*)$$

Proof

(a) Suppose $M[\![(\tau \cup \tau') \cup \tau^*]\!]M'$. Then, by definition $6.a.4$, either $M[\![\tau \cup \tau']\!]M'$ or $M[\![\tau^*]\!]M'$. If $M[\![\tau \cup \tau']\!]M'$, then, again by definition $6.a.4$, either $M[\![\tau]\!]M'$ or $M[\![\tau']\!]M'$. From $M[\![\tau']\!]M'$ and $M[\![\tau^*]\!]M'$ follows $M[\![\tau' \cup \tau^*]\!]M'$. From $M[\![\tau]\!]M'$ and $M[\![\tau' \cup \tau^*]\!]M'$ follows $M[\![\tau \cup (\tau' \cup \tau^*)]\!]M'$. ■

(b) Similar to (a), by decomposing and recomposing according to definition $6.a.3$. ■

The following will be obvious and need no proof, as it follows immediately from the constructions of agent access in clauses $6.a.1$, $6.e.2$, and $6.e.3$ of definition 6:

Fact 1 (S5 invariance)
The class of knowledge models is closed under execution of action types.

The model functional action types have a functional interpretation on the level of models, and the deterministic action types have a functional interpretation on the level of worlds:

Proposition 3 (Functional interpretation)
Let M be a knowledge model, let $w \in M$.

$$(a) \quad \forall \tau \in \mathsf{FuncKT} : \tau \text{ is executable on } M \Rightarrow \exists ! M' : M[\![\tau]\!]M'$$

$$(b) \quad \forall \tau \in \mathsf{DetKT} : \tau \text{ is executable on } (M, w) \Rightarrow \exists !(M', w') : (M, w)[\![\tau]\!](M', w')$$

Proof

(a) Immediate, by induction on τ. The only interesting case is $L_B\tau$, where $\tau \in \mathsf{KT}$ and therefore τ may be not model functional. Suffices to observe that definition 6 constructs a single model $\bigoplus_B [M[\![\tau]\!]]$, regardless of the number of models $[M[\![\tau]\!]]$ contains.

(b) Immediate, by induction on τ.

Proposition 4 (Bisimilarity invariance)
Let M, M' be $S5$ models, and $M \underline{\leftrightarrow} M'$ (M is bisimilar to M'), let $\mathfrak{R}$ be the bisimulation between M and M', and let $\tau \in \mathsf{KT}$. Then:

$$\forall M^* : M[\![\tau]\!]M^* \Rightarrow \exists M^\bullet : (M'[\![\tau]\!]M^\bullet \text{ and } M^* \underline{\leftrightarrow} M^\bullet \text{ and (i) })$$

(i): For the bisimulation $\mathfrak{R}^\tau$ between M^* and $M^\bullet$ it holds that:

$$\forall w^* \in M^*, \forall w^\bullet \in M^\bullet : w^* \mathfrak{R}^\tau w^\bullet \Rightarrow [\![\tau]\!]^{-1}(w^*) \mathfrak{R} [\![\tau]\!]^{-1}(w^\bullet)$$

Proof: By induction on the structure of action types. The most interesting case is $L_B\pi$. The proof is found in the appendix.

Directly from proposition 4 follows the next (corollary, but because of its importance) proposition:

Proposition 5
If M, M' are $S5$ models, and $M \underline{\leftrightarrow} M'$, and $\tau \in \mathsf{FuncKT}$, then $M[\![\tau]\!] \underline{\leftrightarrow} M'[\![\tau]\!]$.

4 Knowledge actions

We now continue by defining the class of knowledge actions. Remember example 2, the action 'player 1 shows (only) player 2 the red card' and its corresponding type 'player 1 shows (only) player 2 his card'. Although we have now formally described and interpreted this action's type, we still cannot describe the action itself. In this section, we extend our language with the operation of 'local choice', to make that possible.

A knowledge action can be formed from an action type τ by a mapping operation $!_I\tau$, where $!_I$ determines *local choice* in τ. Index I determines a subtree in the structural tree of action type τ.

The *labeled structure* operation ls maps each action type to the tree underlying its structural tree as follows: all nodes that are branching due to nondetermistic choice are labeled with different variables; all other nodes and all arcs are not labeled. A variable corresponds to a local choice for a subgroup, as we are about to see.

Definition 10 (Labeled structure of an action type)
Assume a (countably) infinite supply of variables X.

$$\begin{array}{rcl} ls(?\varphi) & = & () \\ ls(L_B\tau) & = & (ls(\tau)) \\ ls(\tau \cup \tau') & = & x(ls(\tau), ls(\tau')) \text{ for some fresh variable } x \\ ls(\tau \ ; \ \tau') & = & (ls(\tau), ls(\tau')) \end{array}$$

Bundle

Given the structure $ls(\tau)$ of an action type, a valuation $val : X \to \mathbb{B}$ (we may as well, and will, write $val \in \mathbb{B}^{\mathbb{N}}$) of the variables in its labeled structure defines a rooted subtree $val(ls(\tau))$ that we call a *bundle*.[5],[6] Value 0 selects the 'left' arc of the current subtree, whereas 1 selects the 'right' arc. By overloading the notation of val we can map a labeled structure to a bundle. A crucial clause in that mapping is: $val(ls(\tau \cup \tau')) = val(x(ls(\tau), ls(\tau'))) = val(x)(val(ls(\tau)), val(ls(\tau')))$. If $val(x) = 1$, this is $1(val(ls(\tau)), val(ls(\tau')))$, if $val(x) = 0$, this is $0(val(ls(\tau)), val(ls(\tau')))$. A bundle can be more than a branch. For a sequential execution both arcs will be part of the bundle. Because of that we call it a bundle: it's not (necessarily) a branch, it's not (necessarily) the entire tree, it's something in between: just a couple of branches.

For convenience, we now define $bu(\tau) = \{val(ls(\tau)) \mid val \in \mathbb{B}^{\mathbb{N}}\}$; $bu(\tau)$ is the set of (different) bundles of τ. Obviously, we do not require that two different valuations define different bundles. We do not even require that two different bundles define different actions, as will become clear from the examples, below.

Each bundle in the structural tree of an action type defines a knowledge action: if $\tau \in \mathsf{KT}$ and $I \in bu(\tau)$, then $!_I\tau$ defines a knowledge action. KA is the class of all syntactic objects thus formed. Write α for an arbitrary knowledge action.

Definition 11 (Knowledge actions – KA)
Let $\mathbf{A}$ be a set of agents (the public), let $\mathbf{P}$ be a set of atoms.

$$\mathsf{KA} = \{!_I\tau \mid \tau \in \mathsf{KT} \text{ and } I \in bu(\tau)\}$$

The modal agents and the group of an action are those of its type, see definitions 15 and 14. The relation between knowledge action types and knowledge actions is the following: to each type corresponds a set of actions (its 'instances',

[5]In order to distinguish val from valuations V_w on models, we have chosen a different notation.

[6]We copy the similar use of the term 'bundle' from modal sequent calculus.

the 'actions of that type'); each action 'is of a certain type'. In a way, deterministic action types *are* knowledge actions, as $\tau \in \mathsf{DetKT} \Rightarrow \forall I \in bu(\tau) : \; !_I\tau = \tau$; in that case, $bu(\tau)$ consists of a single bundle ('no choice'). This will be made precise in section 4.4.

The formal notation with bundles prefixed to action types is a bit cumbersome. The following defines a more convenient notation for knowledge actions.

Definition 12 (Notational equivalent for knowledge actions)

$$\begin{array}{rcl} !_{()}(?\varphi) & := & !?\varphi \\ !_{(I)}(L_B\tau) & := & !L_B \; !_I\tau \\ !_{0(I,J)}(\tau \cup \tau') & := & !_I\tau \cup \tau' \\ !_{1(I,J)}(\tau \cup \tau') & := & \tau \cup \; !_J\tau' \\ !_{(I,J)}(\tau \; ; \; \tau') & := & !_I\tau \; ; \; !_J\tau' \end{array}$$

The only position where an exclamation mark '!' meaningfully remains, is just before a learning operator, i.e. in a subprogram of the form $!L_B\pi$. This 'means' (see below) that only the agents in B know that they have selected π for execution, from the alternatives. The '!' in front of a test $!?\varphi$ has no meaning, see the example in the next subsection.

Further simplifications: If there is nothing to choose from, we can delete the exclamation mark. That means we only have to keep '!'-s in subprograms of the form $!\pi \cup \pi'$ or $\pi \cup \; !\pi'$.

Warning: the local interpretation of actions, to be defined in the next subsection, is only compositional from the viewpoint of formal notation, not from that of its notational equivalent.

We continue with some examples.

4.1 Examples

We are now ready to describe the examples from the introductory section 1. From subsection 3.2 we already know their types.

Example 1
The first example, 'player 1 puts the red card on the table', is not of much interest. Its structural tree is a branch, which (therefore) also is its only bundle: $ls(L_{123}?r_1) = (())$. The next example we explain in detail:

Example 2
This is the action 'player 1 shows (only) player 2 the red card'. The type of this action is: 'player 1 shows his card to player 2', formally $L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$.

To illustrate the construction of an action of this type, we have to be more precise about its structure. We can – arbitrarily, as $\cup$ is associative – take the action type to be:

$$L_{123}((L_{12}?r_1 \cup L_{12}?w_1) \cup L_{12}?b_1).$$

The labeled structure of this action type is:

$$(x(y(((()),(())),(())))).$$

The labeled structure consists of three bundles:

- $(0(0(((()),(())),(()))))$
- $(0(1(((()),(())),(()))))$
- $(1(0(((()),(())),(()))))$

Note that $(1(0(((()),(())),(()))))$ defines the same bundle as $(1(1(((()),(())),(()))))$: once we have chosen the *right* subtree, we don't care what choices are made further down in the pruned *left* subtree.

Figure 8 illustrates how the middle one of the three bundles is created. On the left in the figure, the labeled structure $ls(L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1))$. In the middle, the bundle $val(ls(L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)))$, for the valuation $val(x) = 0$ and $val(y) = 1$. On the right in the figure, the actual bundle that is selected by this formally defined bundle.

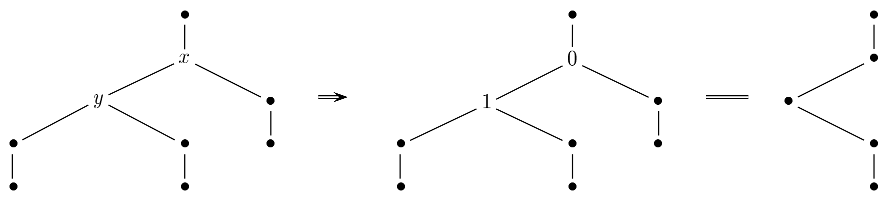

Figure 8: How to make a bundle from a tree

The three different bundles result in the three different actions:

- $!L_{123}((!L_{12}!?r_1 \cup L_{12}!?w_1) \cup L_{12}!?b_1)$
- $!L_{123}((L_{12}!?r_1 \cup \ !L_{12}!?w_1) \cup L_{12}!?b_1)$
- $!L_{123}((L_{12}!?r_1 \cup L_{12}!?w_1) \cup \ !L_{12}!?b_1)$

With the simplifications mentioned before, and deleting some superfluous brackets, because nondeterministic choice is associative, we get:

- $L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$
- $L_{123}(L_{12}?r_1 \cup \ !L_{12}?w_1 \cup L_{12}?b_1)$
- $L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup \ !L_{12}?b_1)$

The first of these, $L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$, expresses the action 'player 1 shows (only) player 2 the red card'. The exclamation mark '!' in front of the subprogram $L_{12}?r_1$ means: from the three available alternatives, agents 1 and 2 choose $L_{12}?r_1$. This choice is local, i.e. known to 1 and 2 *only*. Player 3 doesn't know which of the three alternatives has been chosen, although he knows what to choose from.

Example 3
'Player 2 asks player 1 for the white card; player 1 says that he doesn't have it; player 2 ends his move.' The type of this action is $L_{123}?\neg w_1 ; L_{123}?\neg \mathsf{win}_2$. The labeled structure of this action type contains no variables. The only bundle in its structural tree is the tree itself. Therefore the only action corresponding to it is the action type itself. Note that the bundle, unlike those in the previous examples, is *not* a branch.

Example 4
The action is 'player 1 whispers in 2's ear that he doesn't have the white card'. Its type is 'player 1 whispers in 2's ear a card that he (1) doesn't have', formally: $L_{123}(L_{12}?\neg r_1 \cup L_{12}?\neg w_1 \cup L_{12}?\neg b_1)$. It will now be clear that the KA action describing example 4 is:

$$L_{123}(L_{12}?\neg r_1 \cup \ !L_{12}?\neg w_1 \cup L_{12}?\neg b_1)$$

Choosing between tests

We end by explaining the remark after definition 12, on the meaningless of choosing between tests. Consider the action type $\chi = L_{123}(?r_1 \cup ?w_1)$. Observe that $ls(\chi) = (x((),()))$. The structural tree therefore contains two different bundles, $I = (0((),()))$, and $J = (1((),()))$. E.g. $!_I\chi$ is the action where everybody in $gr(\chi) \setminus gr(?r_1)$ cannot distinguish between the tests $?r_1$ and $?w_1$, whereas everybody in $gr(?r_1)$ learns that $?r_1$ has really been executed. Computing the groups: *everybody* learns that $?r_1 \cup ?w_1$ whereas *nobody* learns that $?r_1$. Therefore (as in the previous example, but for a different reason): $!_I\chi = \chi$. Of course, similarly $!_J\chi = \chi$. Indeed, although it contains a $\cup$ operator, the action type χ is deterministic, and the two different bundles I and J define the same action.

In simplified notation, we can *always* delete '!' in front of tests, because 'local choice' between two tests corresponds to 'nobody' learning which of those tests is chosen.

4.2 Local interpretation of knowledge actions

The local interpretation of a knowledge action is a functional relation $[\![\cdot]\!]$ between knowledge states. Again, we overload the $[\![\cdot]\!]$ notation, to include the interpretation of actions. To interpret a knowledge action $\alpha = \ !_I\tau$ on a knowledge state (M, w) we first determine the interpretation of τ on M. From the set $[M[\![\tau]\!]]$ we then select *one* model M' and in that model M' *one* world w'.

Both are truly selections, as $[M[\![\tau]\!]]$ can contain more than one model, and M' can contain more than one τ-image of w. What M' and w' are, is determined by the bundle I. The world w' is the $!_I\tau$ image of w, to be defined below as a relation $\mapsto_{!_I\tau}$, and M' is simply the model that contains w'. (M', w') is then the required knowledge state.

Definition 13 (Local interpretation of knowledge actions)
Let $(M, w) = (\langle W, (\sim_a)_{a\in A}, V\rangle, w)$ be a knowledge state. Let τ be an A action type. Let I be a bundle in $ls(\tau)$.

13.a.1	$(M, w)[\![!_I\tau]\!](M', w')$	iff	$M[\![\tau]\!]M'$ and $w \mapsto_{!_I\tau} w'$
13.b			$\forall M, M' : M[\![\tau]\!]M', \forall w \in M, \forall w' \in M' :$
13.b.1	$w \mapsto_{!_{()}?\varphi} w'$	iff	$w = w'$
13.b.2	$w \mapsto_{!_{(I)}L_B\tau} w'$	iff	$\exists M'' \in [M[\![\tau]\!]] : \exists w'' \in M'' :$ $w' = (M'', w'')$ and $w \mapsto_{!_I\tau} w''$
13.b.3	$w \mapsto_{!_{(I,J)}(\tau\ ;\ \tau')} w'$	iff	$\exists M'' : M[\![\tau]\!]M''[\![\tau']\!]M', \exists w'' \in M'' :$ $w \mapsto_{!_I\tau} w'' \mapsto_{!_J\tau'} w'$
13.b.4	$w \mapsto_{!_{0(I,J)}(\tau\cup\tau')} w'$	iff	$w \mapsto_{!_I\tau} w'$
13.b.5	$w \mapsto_{!_{1(I,J)}(\tau\cup\tau')} w'$	iff	$w \mapsto_{!_J\tau'} w'$

Because the local interpretation of an action is functional on every knowledge state, instead of $(M, w)[\![!_I\tau]\!](M', w')$ we write $(M', w') = (M, w)[\![!_I\tau]\!]$. We say that $(M, w)[\![!_I\tau]\!]$ is *the* interpretation of $!_I\tau$ in (M, w). As before, we *might* even write $(M[\![!_I\tau]\!], w[\![!_I\tau]\!])$. However, only the type of an action determines the structural changes in the model. Therefore, if τ is model functional, instead of $(M[\![!_I\tau]\!], w[\![!_I\tau]\!])$ we *will* write $(M[\![\tau]\!], w[\![!_I\tau]\!])$. Of course, $w[\![!_I\tau]\!]$ is precisely the $\mapsto_{!_I\tau}$-image of w.

We have chosen the $\mapsto$ symbol for its functional connotation. Note that $w \to_\tau w'$, as in definition 6.c, relates w to one of its τ-images, whereas $w \mapsto_{!_I\tau} w'$ relates w to its unique $!_I\tau$-image.

Warning: although the semantics of knowledge actions is compositional, it *appears* to be not compositional when we use the notational equivalent from definition 12. For an example, the interpretation of $!L_{12}?r_1$ in $L_{123}(!L_{12}?r_1 \cup L_{12}?b_1)$ is *not* a function of the interpretation of '!', whatever that might mean, and $L_{12}?r_1$. In the next subsection we give an example of how to apply definition 13.

The definitions of 'executable', 'equivalence', and 'public interpretation' are extended to include actions. See definitions 7, 8, and 9:

A knowledge action $!_I\tau$ is *executable* in a knowledge state (M, w), if the local interpretation of $!_I\tau$ in (M, w) is not the empty relation.

Two KA actions $!_I\tau$ and $!_J\tau'$ are *equivalent*, notation $!_I\tau =!_J\tau'$, if they have the same interpretation: i.e. if $[\![!_I\tau]\!] = [\![!_J\tau']\!]$. We even have: Two $\mathsf{KT} \cup \mathsf{KA}$ programs π and π' are *equivalent*, if $[\![\pi]\!] = [\![\pi']\!]$.

If $gr(\alpha) = gr(M)$ then $[\![\alpha]\!]$ is the *public* interpretation of α in M.

4.3 Examples

To illustrate the semantics of actions we compute the (public) interpretation of knowledge action $L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$ in (hexa, rwb).

$(\mathsf{hexa}, rwb)[\![L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)]\!](M', w')$
$\Leftrightarrow$ definition $13.a.1$
$\mathsf{hexa}[\![L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)]\!]M'$ and
$rwb \mapsto_{L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)} w'$

The model $M' = \mathsf{hexa}[\![L_{123}(L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)]\!]$ is computed in example 2 (see figure 3). The image of rwb in that model is computed as follows (better read the computation bottom-up):

$rwb \mapsto_{L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)} (\mathsf{hexa}[\![L_{12}?r_1]\!], (\mathsf{hexa}[\![?r_1]\!], rwb))$
$\Leftrightarrow$ definition $13.b.2$
$rwb \mapsto_{!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1} (\mathsf{hexa}[\![?r_1]\!], rwb)$
$\Leftrightarrow$ definition $13.b.4$, twice
$rwb \mapsto_{L_{12}?r_1} (\mathsf{hexa}[\![?r_1]\!], rwb)$
$\Leftrightarrow$ definition $13.b.2$
$rwb \mapsto_{?r_1} rwb$
$\Leftrightarrow$ definition $13.b.1$
$rwb = rwb$

If we write unions instead of direct sums for domains, we simply get rwb, instead of $(\mathsf{hexa}[\![L_{12}?r_1]\!], (\mathsf{hexa}[\![?r_1]\!], rwb))$, as the required image of rwb in $\mathsf{hexa}[\![L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)]\!]$. See also figure 9, to be compared with figure 3.

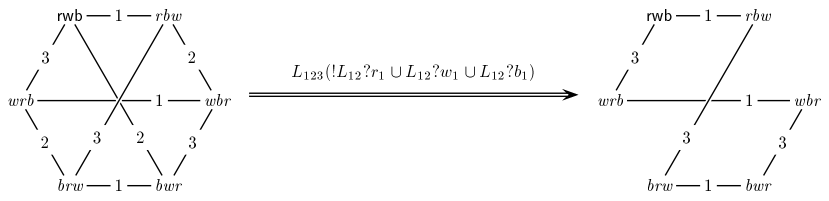

Figure 9: Interpreting action $L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1)$ in state $(\mathsf{hexa}, \mathsf{rwb})$

4.4 Action properties

In this subsection we prove some elementary action properties.

Proposition 6 (Relation between actions and action types)

$$\begin{array}{ll} (a) & (M,w)[\![!_I\tau]\!](M',w') \Rightarrow M[\![\tau]\!]M' \\ (b) & (M,w)[\![!_I\tau]\!](M',w') \Rightarrow (M,w)[\![\tau]\!](M',w') \\ (c) & (M,w)[\![\tau]\!](M',w') \Rightarrow \exists I \in bu(\tau) : (M,w)[\![!_I\tau]\!](M',w') \end{array}$$

Proof

(a) By definition. ■

(b) Suffices to observe that very world $w' \in M' \in [[\![\tau]\!]]$ has a unique origin $[\![\tau]\!]^{-1}(w')$ in M, so that $w = [\![\tau]\!]^{-1}(w')$. ■

(c) We construct a bundle for τ using definition 13. Instead of (c) we prove: $(M,w)[\![\tau]\!](M',w') \Rightarrow \exists I \in bu(\tau) : w \mapsto_{!_I\tau} w'$, by induction on τ; (c) then follows immediately, from definition 13. A typical case: Let $(M,w)[\![\tau \cup \tau']\!](M',w')$. Then either $(M,w)[\![\tau]\!](M',w')$ or $(M,w)[\![\tau']\!](M',w')$. Suppose $(M,w)[\![\tau]\!](M',w')$. By induction $\exists I \in bu(\tau)$ such that $w \mapsto_{!_I\tau} w'$. For any $J \in bu(\tau')$, bundle $0(I,J)$ is the required bundle, as $w \mapsto_{!_{0(I,J)}\tau\cup\tau'} w' \Leftrightarrow w \mapsto_{!_I\tau} w'$. ■

Using propositions 1.a and 6.c, we also have, that if a type can be executed on a model, there is at least one executable action of that type: If τ is executable on M, there is a bundle $I \in bu(\tau)$, and a world $w \in M$, such that $!_I\tau$ is *executable* in state (M,w).

Fact 2
Actions are deterministic programs

Obvious.

Fact 3 (Embedding of DetKT into KA)
If $\tau \in \mathsf{DetKT}$, and M, M' knowledge models, then: $\forall I \in bu(\tau) : [\![!_I\tau]\!] = [\![\tau]\!]$.

Using proposition 6.b and 6.c, this will be obvious. The structure of a deterministic action type is an unlabeled tree, that also is its unique bundle. There is therefore only one action instance of a deterministic action type. This is what we mean by deterministic action types being the same as their corresponding actions.

Not to every action corresponds a deterministic action type! A counterexample is the action of example 2: $L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?w_1)$.

Warning: Observe that the class of actions KA is *not* closed under action type constructing operations (assuming that we give them the obvious interpretation). For an example:

$$L_{123}(!L_{12}?r_1 \cup L_{12}?w_1 \cup L_{12}?b_1) \cup L_{123}(L_{12}?r_1 \cup \ !L_{12}?w_1 \cup L_{12}?b_1)$$

is *neither* an action, nor an action type. (In particular, observe that it is not equivalent to the action type $L_{123}(L_{12}?r_1 \cup L_{12}?w_1)$.)

Another warning, relating to proposition 6: that an action type is executable in a model, does not imply that every action of that type, is executable in one of the states of that model. A simple counterexample: the action $L_{123}(L_{12}?r_1 \cup\ !L_{12}?\bot)$ is not executable in any of the states (hexa, w).

5 Extensions of KT and KA, and limitations

The program class KT of action types might be extended in two directions: by adding the operation of simultaneous execution, or by relaxing the constraints on groups in KT formation, as in definition 3. We would also like to interpret 'non-uniform' types, where programs of different subgroups can be combined. We discuss this by means of examples.

5.1 Simultaneous execution

Recall example 5 from the introduction:

Example 5
There are three players (1,2,3) and four cards north, east, south and west (n, e, s, w). Player 1 holds north, player 2 east and player 3 south and west. Player 3 shows his south card (only) to player 1, with his left hand, and (simultaneously) his west card (only) to player 2, with his right hand.

We add a new action type construction operator: simultaneous execution $\cap$ (e.g. see [Gol92]); $\tau \cap \tau'$ is the action type where τ and τ' are executed simultaneously.

The 'knowledge action' in example 5 can be described more precise as: 'The public (1, 2, and 3) learn that 3 shows simultaneously a card to 1 and another card to 2, and the alternative actually chosen is that 3 shows south to 1 and west to 2'. This is expressed by:

$$L_{123}(!(L_{13}?s_3 \cap L_{23}?w_3) \cup \bigcup_{i \neq j, (i,j) \neq (s,w)} (L_{13}?i_3 \cap L_{23}?j_3))$$

The 'action type' corresponding to it, should be:

$$L_{123}(\bigcup_{i \neq j}(L_{13}?i_3 \cap L_{23}?j_3))$$

It is clear, what knowledge state describes the initial dealing of cards, and what unique knowledge state results from the execution of this action (by checking expected postconditions in that state), see [vD00c].

We have two technical complications: 'simultaneous execution' and the combination of subprograms for different groups: 'nonuniform types' (in definition 3 we required that $gr(\tau) = gr(\tau')$, in constructs $\pi \cup \pi$ and $\pi\ ;\ \pi$). As we do not know how to interpret it, we have not incorporated simultaneous execution in our framework.

But do we really need it? It is instructive to understand *why* example 5 cannot be modelled as a KA action. The direction in which we have to look, is to remodel it as a sequence of actions or action types where a single card is shown:

The following is a KT action type:

$$L_{123}(L_{13}?n_3 \cup L_{13}?e_3 \cup L_{13}?s_3 \cup L_{13}?w_3)$$

We abbreviate it as ('3 shows a card to 1'):

$$\mathsf{show}^{1,-}_{3,-}$$

The following is a KA action:

$$L_{123}(L_{13}?n_3 \cup L_{13}?e_3 \cup \ !L_{13}?s_3 \cup L_{13}?w_3)$$

We abbreviate this action as ('3 shows the south card to 1'):

$$\mathsf{show}^{1,-}_{3,s}$$

The following is *still* a KA action, as $gr(\mathsf{show}^{1,-}_{3,s}) = gr(\mathsf{show}^{2,-}_{3,w}) = \{1, 2, 3\}$:

$$\mathsf{show}^{1,-}_{3,s}\,;\mathsf{show}^{2,-}_{3,w}$$

However, its type (and therefore: what its model looks like) is:

$$\mathsf{show}^{1,-}_{3,-}\,;\mathsf{show}^{2,-}_{3,-}$$

Unfortunately, but not surprisingly, the model resulting from executing that type is more complex than we want. This is because, obviously, once player 3 has shown a card, nothing prevents him showing the same card *again* to another player: more possibilities, more worlds.

Showing different cards to different players *seems* to correspond to:

$$\bigcup_{i \neq j \in \{n,e,s,w\}} (\mathsf{show}^{1,-}_{3,i}\,;\mathsf{show}^{2,-}_{3,j})$$

However, knowledge actions are not closed under program constructing operations, so this is *not* an action. (So what? But what would be its interpretation? Note that a unique model is supposed to be produced, not a set of models.)

There seems another line of approach to model example 5, that is also instructive. Given that we cannot describe two different cards being simultaneously shown, why not somehow include the *effect* of one card being shown in the preconditions of the action where the other card is shown, or in a separate, third action in between or after those two? This is the difference between examples 5 and 6, and brings us to the next subsection.

5.2 Epistemic and other aspects of actions

Example 6
Player 3 shows his south card (only) to player 1; next, player 2 asks player 3 to show him his other card; player 3 shows his west card (only) to player 2

A consequence of 3 showing 1 the south card, is that 1 knows that card. We include that condition in the test of the second action, where 3 shows the other card, the west card, to 2. Its type would then be:

$$L_{123}(L_{23}?(n_3 \wedge \neg K_1 n_3) \cup L_{23}?(e_3 \wedge \neg K_1 e_3) \cup L_{23}?(s_3 \wedge \neg K_1 s_3) \cup L_{23}?(w_3 \wedge \neg K_1 w_3))$$

The action type describes that 3 shows player 2 a card that player 1 doesn't know. Given an initial dealing of cards where 3 holds south and west, and where the first action was of type $L_{123}(L_{13}?n_3 \cup L_{13}?e_3 \cup L_{13}?s_3 \cup L_{13}?w_3)$, the knowledge state resulting from executing this action is indeed the intended knowledge state! This even holds for *any* initial dealing of cards. However, its interpretation is not the same on all knowledge states: if player 1 already knew all cards, the interpretation of the action above is empty, whereas player 3 can still show both cards to different players: that action's interpretation is not empty. It's just that player 1 doesn't gain any knowledge from that transaction.

It doesn't help to strengthen the test, and replace $n_3 \wedge \neg K_1 n_3$, and so on, by the strongest postcondition $n_3 \wedge \neg C_{13} n_3$, or by $n_3 \wedge [\alpha]\varphi$, for some suitable KA action α and DKL formula φ. This is because of an essential limitation of our approach: we can only refer to the epistemic effects of actions and their types, not to other action features, such as when they took place. In particular, there is no way to refer to the *previous* action, which is what is expressed in example 6.

So we seem be beyond the expressive power of our language.

This also ends the discussion on simultaneous execution, an operation that we hope to extend our language with. We have some separate remarks on nonuniform types, again for an instructive example.

5.3 Nonuniform programs

Example 5 contained the simultaneous execution of programs for different subgroups. We now look at some examples for choice between programs for different subgroups. We call them nonuniform programs. When at some level the public learns about these alternatives, there often seems to be a KT action type or KA action that is *equivalent* to that nonuniform program. Equivalent means: having the same interpretation, where the nonuniform programs are interpreted by considering their postconditions, formulated in terms of subgroup common knowledge.

$$\begin{array}{rcl}
L_{123}(L_{12}?r_1 \cup L_{23}?r_1) & \text{is equivalent to} & L_{123}?r_1 \\
L_{123}(L_{12}?r_1 \cup L_{23}?w_1) & \text{is equivalent to} & L_{123}?r_1 \cup L_{123}?w_1 \\
L_{123}(L_2?r_1 \cup L_3?r_1) & \text{is equivalent to} & L_{123}?r_1 \\
L_{123}(L_2?r_1 \cup L_3?w_1) & \text{is equivalent to} & L_{123}(L_{23}?r_1 \cup L_{23}?w_1)
\end{array}$$

Apparently we can relax some of the constraints on groups in the construction of action types in definition 3. In section 3.7 we already mentioned that the constraint $gr(\tau) \subset B$ for $L_B\tau$ can be relaxed.

We then can prove general program properties such as (let $B, C \subseteq A$):

$$\begin{array}{rcl}
L_A(L_B\tau \cup L_C\tau') & = & L_A(L_{B\cup C}\tau \cup L_{B\cup C}\tau') \\
L_A(L_A\tau \cup L_A\tau') & = & L_A\tau \cup L_A\tau'
\end{array}$$

This concludes our diversion into possible extensions of the class KT of action types.

6 Conclusion

Our goal was to define a language for epistemic dynamics in information systems that model knowledge and reason about knowledge: multiagent $S5$ systems. We fullfilled that goal by proposing a logical language DKL, for dynamic knowledge logic, that includes a language KT of action types and a derived language KA of knowledge actions. Basic to our approach is the concept of local interpretation of an action type in a model, the interpretation fo a subgroup of agents only. We performed detailed computations on some example knowledge game actions, to illustrate the language and its interpretation. We discussed possible extensions with simultaneous execution and with combining programs for different groups. We have achieved a small goal in Gerbrandy's research program: define a program class under which application the class of $S5$ models is closed. Contrary to Gerbrandy, an action can be locally interpreted without having to assume that the other agents learn nothing. In [vD00b], in preparation, we compare our approach in detail to that of [Ger99] and that of [Bal99].[7]

Appendix

Definition 14 (Group)
Let $M = \langle W, (\sim_a)_{a\in A}, V\rangle$ be an $S5$ model. Let $\tau \in \mathsf{KT}$. Let $!_I\tau = \alpha \in \mathsf{KA}$. Then:

[7]That includes representing the set of actions of a given type as an $S5$ frame, such that executing a type on a model can be seen as multiplying that frame with that model.

$$\begin{array}{rcl}
gr(M) & = & A \\
gr((M,s)) & = & gr(M) \\
 & & \\
gr(?\varphi) & = & \emptyset \\
gr(L_B\tau) & = & B \cup gr(\tau) \\
gr(\tau\ ;\ \tau') & = & gr(\tau) \cup gr(\tau') \\
gr(\tau \cup \tau') & = & gr(\tau) \cup gr(\tau') \\
 & & \\
gr(!_I\tau) & = & gr(\tau)
\end{array}$$

Definition 15 (Modal agents)
With induction on the structure of formulas $\varphi \in \mathsf{DKL}$, and on the structure of types $\tau \in \mathsf{KT}$, and for all bundles $I \in bu(\tau)$, we define:

$$\begin{array}{rcl}
ag(p) & = & \emptyset \\
ag(\neg\varphi) & = & ag(\varphi) \\
ag(\varphi \wedge \psi) & = & ag(\varphi) \cup ag(\psi) \\
ag(K_a\varphi) & = & ag(\varphi) \cup \{a\} \\
ag(C_B\varphi) & = & ag(\varphi) \cup B \\
ag([\pi]\varphi) & = & ag(\varphi) \cup ag(\pi) \cup gr(\pi) \\
 & & \\
ag(?\varphi) & = & ag(\varphi) \\
ag(L_B\tau) & = & ag(\tau) \\
ag(\tau\ ;\ \tau') & = & ag(\tau) \cup ag(\tau') \\
ag(\tau \cup \tau') & = & ag(\tau) \cup ag(\tau') \\
 & & \\
ag(!_I\tau) & = & ag(\tau)
\end{array}$$

Proof of proposition 4

By induction on the structure of action types.

Case $?\varphi$. Define $\mathfrak{R}^{?\varphi}(w,v) \Leftrightarrow \mathfrak{R}(w,v)$, for all worlds $w \in M[\![?\varphi]\!]$ and $v \in M'[\![?\varphi]\!]$. Note that $[\![?\varphi]\!]^{-1}(w') = w'$ for any world w' from any model resulting from executing φ.

Relation $\mathfrak{R}^{?\varphi}$ is a bisimulation between $M[\![?\varphi]\!]$ and $M'[\![?\varphi]\!]$, because both $M[\![?\varphi]\!]$ and $M'[\![?\varphi]\!]$ are $\emptyset$ models (models without access) and because Define $\mathfrak{R}^{?\varphi}(w,v) \Leftrightarrow \mathfrak{R}(w,v)$, for all worlds $w \in M[\![?\varphi]\!]$ and $v \in M'[\![?\varphi]\!]$, $\mathfrak{R}^{?\varphi}(w,v)$ implies that $\mathfrak{R}(w,v)$, which implies that $V_w = V_v$.

(i): Obviously, we also have that for all worlds $w \in M[\![?\varphi]\!]$ and $v \in M'[\![?\varphi]\!]$: $w\mathfrak{R}^{\varphi}v \Leftrightarrow [\![?\varphi]\!]^{-1}(w)\mathfrak{R}^{\varphi}[\![?\varphi]\!]^{-1}(v)$, because the last is equivalent to: $w\mathfrak{R}v$.

Case $\tau\ ;\ \tau'$. This can be shown directly. Suppose $\tau\ ;\ \tau'$ is executable on M, then τ is executable on M and τ' is executable on $M[\![\tau]\!]$. Now:

If $M \underline{\leftrightarrow} M'$ and $M[\![\tau]\!]M^*$, then by the induction hypothesis there is an $M^\bullet$ such that $M'[\![\tau]\!]M^\bullet$ and $M^* \underline{\leftrightarrow} M^\bullet$. If $M^*[\![\tau']\!]M_1$, then by the induction hypothesis there is an M_2 such that $M^\bullet[\![\tau']\!]M_2$ and $M_1 \underline{\leftrightarrow} M_2$, i.e.: $M[\![\tau]\!][\![\tau']\!]M_1$ and $M'[\![\tau]\!][\![\tau']\!]M_2$. As $[\![\tau]\!][\![\tau']\!] = [\![\tau\ ;\ \tau']\!]$ we are ready.

(i): Obvious. Suppose $\mathfrak{R}^\tau$ is the bisimulation between M^* and $M^\bullet$ and $\mathfrak{R}^{\tau;\tau'}$ is the bisimulation between M_1 and M_2, then for all $w_1 \in M_1$, $w_2 \in M_2$:

$w_1 \mathfrak{R}^{\tau;\tau'} w_2$
$\Rightarrow$ induction hypothesis
$[\![\tau']\!]^{-1}(w_1)\mathfrak{R}^\tau[\![\tau']\!]^{-1}(w_2)$
$\Rightarrow$ induction hypothesis
$[\![\tau]\!]^{-1}([\![\tau']\!]^{-1}(w_1))\mathfrak{R}^\tau[\![\tau]\!]^{-1}([\![\tau']\!]^{-1}(w_2))$
$\Leftrightarrow$
$[\![\tau\ ;\ \tau']\!]^{-1}(w_1)\mathfrak{R}^\tau[\![\tau\ ;\ \tau']\!]^{-1}(w_2)$

Case $\tau \cup \tau'$. For nondeterministic types, it is *not* generally the case that bisimilarity is invariant under their execution, as different choices may be made in executing τ on M and M'. Still, any choice made on M *can* be matched by a similar choice on M'. This is expressed by the following argument:

If $M[\![\tau \cup \tau']\!]M^*$ then by definition 6.*a*.4 $M[\![\tau]\!]M^*$ or $M[\![\tau']\!]M^*$. If $M[\![\tau]\!]M^*$, then by induction, there is an $M^\bullet$ such that $M'[\![\tau]\!]M^\bullet$ and $M^\bullet \underline{\leftrightarrow} M^*$. From $M'[\![\tau]\!]M^\bullet$ follows, again by definition 6.*a*.4, that $M'[\![\tau \cup \tau']\!]M^\bullet$. If $M[\![\tau']\!]M^*$, then by induction, there is an M'' such that $M[\![\tau']\!]M''$ and $M'' \underline{\leftrightarrow} M^*$. From $M'[\![\tau']\!]M''$ follows, again by definition 6.*a*.4, that $M'[\![\tau \cup \tau']\!]M''$.

(i): Obvious. Let $\mathfrak{R}^{\tau\cup\tau'}$ be that bisimulation. Then it's either a bisimulation between M' and $M^\bullet$ or one between M' and M''. Use induction either for τ or for τ'.

Case $L_B\tau$. We proceed as follows: execution of τ on M results in a set $[M[\![\tau]\!]]$ of models. By using the induction hypothesis, we may assume that every model from that set is bisimilar to a model in the set $[M'[\![\tau]\!]]$, and vice versa. Therefore, obviously the direct sum $\bigoplus[M[\![\tau]\!]]$ is bisimilar to the direct sum $\bigoplus[M'[\![\tau]\!]]$. Let $\mathfrak{R}^\oplus$ be that bisimulation. In order to compute $M[\![L_B\tau]\!] = \bigoplus_B[M[\![\tau]\!]]$ we still have to add access to $\bigoplus[M[\![\tau]\!]]$ for the agents in $B \setminus gr(\tau)$. We show that the relation $\mathfrak{R}^{L_B\tau} := \mathfrak{R}^\oplus$ defines a bisimulation between $M[\![L_B\tau]\!]$ and $M'[\![L_B\tau]\!]$:

For the agents in $gr(\tau)$ this is obvious. For agents $b \in B \setminus gr(\tau)$:

Let $\mathfrak{R}^{L_B\tau}(w,v)$ and $w \sim_b w'$. There is a $v' \in M[\![L_B\tau]\!]$ such that $\mathfrak{R}^\oplus(w',v')$. We show that $v \sim_b v'$. It holds that:

Suppose $\mathfrak{R}^{L_B\tau}(w,v)$, i.e. $\mathfrak{R}^\oplus(w,v)$. By induction, for all $M^* \in M[\![\tau]\!]$ there is a $M^\bullet \in M'[\![\tau]\!]$ such that $M^* \underline{\leftrightarrow} M^\bullet$. Let $\mathfrak{R}^\tau$ be that bisimulation. By that same induction step, if $\mathfrak{R}^\tau(w,v)$ then $\mathfrak{R}([\![\tau]\!]^{-1}(w), [\![\tau]\!]^{-1}(v))$. Similarly, for w' and v'. We now continue as follows:

$w \sim_b w'$
$\Leftrightarrow$ by definition 6.*e*.3

$[\![\tau]\!]^{-1}(w) \sim_b [\![\tau]\!]^{-1}(w')$
$\Leftrightarrow$ as $\mathfrak{R}$ is a bisimulation between M and M'
$[\![\tau]\!]^{-1}(v) \sim_b [\![\tau]\!]^{-1}(v')$
$\Leftrightarrow$ by definition $6.e.3$
$v \sim_b v'$

This ends the proof of proposition 4. ■

References

[Bal99] Alexandru Baltag. A logic of epistemic actions. Manuscript, 1999.

[BMS98] Alexandru Baltag, Lawrence S. Moss, and Slawomir Solecki. The logic of public announcements and common knowledge. In *Proceedings of TARK 98 conference*, 1998.

[FHMV95] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge MA, 1995.

[Ger99] Jelle Gerbrandy. *Bisimulations on Planet Kripke.* PhD thesis, University of Amsterdam, Amsterdam, 1999. ILLC Dissertation Series DS-1999-01.

[Gol92] Robert Goldblatt. *Logics of time and computation.* CSLI Publications, Stanford CA, 2 edition, 1992. CSLI Lecture Notes No. 7.

[Har84] David Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604, Dordrecht, 1984. Kluwer Academic Publishers.

[Koo99] Barteld Kooi. The monty hall dilemma. Master's thesis, Groningen University, Groningen, the Netherlands, 1999.

[MvdH95] J.-J.Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science.* Cambridge Tracts in Theoretical Computer Science 41. Cambridge University Press, Cambridge MA, 1995.

[Ren99] G.R. Renardel de Lavalette. Memories and knowledge games. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema, editors, *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*, Amsterdam, 1999. Amsterdam University Press.

[vD99] Hans van Ditmarsch. The logic of knowledge games: showing a card. In Eric Postma and Marc Gyssens, editors, *Proceedings of the Netherlands/Belgium Conference on Artificial Intelligence (BNAIC 99)*, pages 35–42, Maastricht University, 1999.

[vD00a] Hans van Ditmarsch. Axioms for knowledge games. Manuscript to be included in PhD thesis, 2000.

[vD00b] Hans van Ditmarsch. Comparison to other approaches. Manuscript to be included in PhD thesis, 2000.

[vD00c] Hans van Ditmarsch. Examples. Manuscript to be included in PhD thesis, 2000.

[vD00d] Hans van Ditmarsch. Knowledge games. Manuscript to be included in PhD thesis, 2000.