



UvA-DARE (Digital Academic Repository)

Epistemic modelling and protocol dynamics

Wang, Y.

Publication date
2010

[Link to publication](#)

Citation for published version (APA):

Wang, Y. (2010). *Epistemic modelling and protocol dynamics*. [Thesis, fully internal, Universiteit van Amsterdam]. Institute for Logic, Language and Computation.

General rights

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

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

Chapter 4

Logics of Knowledge and Protocol Change

4.1 Introduction

In Chapter 3 we have shown that knowing that an epistemic protocol would fulfil a certain goal may affect the verification of the protocol. In this chapter, we draw the attention to the knowledge and the dynamics of protocols. As we motivated in Chapter 1, knowing a protocol means *knowing what to do* [HF89] and *knowing the meaning carried by actions* according to the protocols [PR03]. In this chapter, we will make these two observations more precise. More importantly, we address the problem “how to know a protocol?” by modelling the dynamics of protocols.

Knowing what to do In the framework L_{AP} of the previous chapter, a promising candidate for expressing that an agent knows what to do according to an announcement protocol π is the formula $K_i\langle\pi\rangle\top$. However, due to the semantics of L_{AP} , $K_i\langle\pi\rangle\top$ only says that agent i knows that π can be executed at any world he considers possible according to the inherent preconditions of the announcements in π . For example, in the Muddy Children scenario (Example 1.1.2) the assumed protocol is to (repeatedly) announce whether you yourself are muddy, and clearly you know you can announce it. However, there are many other true propositions that could be announced by an agent. For example, $K_i\langle!_i m_j + !_i \neg m_j\rangle\top$ is valid in the model for $i \neq j$, but $!_i m_j + !_i \neg m_j$ is not part of the intended protocol. Clearly, we need a constraint telling us which announcements are in accordance with the protocol, in other words, we need to model the role of the *father* as in the original story of the Muddy Children .

The existing work on protocols in DEL enriches the epistemic models with explicit protocols such that the possible behaviours of agents are not only restricted by the inherent preconditions of epistemic events but also restricted by protocol information [HY09, vBGHP09, Hos09, HP10b]. This is similar to the treatment of protocols in ETL [HF89, PR03], where the temporal development of a system is generated from an initial situation by a commonly known protocol¹. In this chapter we take a different approach: we precisely model the role of the father as in the Muddy Children

¹However, the framework in [vBGHP09] can also handle the protocols which are not common known.

scenario by introducing *protocol announcements* $[!\pi]$ in the language. For example, we use $[!(a \cdot b)]-\langle b \rangle \phi$ to express: after the announcement of the protocol $a \cdot b$, b can not be executed as the first event². The semantics of the language with protocol announcements is defined on standard Kripke models. The extra protocol information is only introduced by protocol announcements while evaluating a formula. Such an approach makes it possible to not only model the “installation” of the initial protocol explicitly but also to handle protocol changes during the execution of the current protocol: we model a true father who may change his mind.

The dynamics of protocols often occur in social interactions. For example, imagine that you were told to close the door and on your way to do it you are told again not to close it. Also as we mentioned in Chapter 1, someone from France may need to update his protocol on cheek kissing when living in Holland. As another example, let consider the yes-no questions which can be viewed as protocols announced by the questioner: answer “Yes!” or answer “No!”. In dialogues, a well-trained spokesman may respond to a yes-no question by inserting yet another protocol: “before answering your question, tell me what you meant by ϕ .”

Knowing what the actions mean The dynamics of the protocols that carry meanings for actions are even more interesting. Here is yet another example: the Chinese are *non-confrontational* in the sense that they will not overtly say “no”, instead they say “I will think about it” or “we will see”. For a western businessman, “we will see”, according to the standard interpretation, means it is still possible. However, if he is updated with the Chinese protocol: $?p_{no} \cdot a_{will-see}$ then he should see this is just another way of saying “No”. Note that in the standard DEL, the interpretations of events are fixed and implicitly assumed to be common knowledge, e.g., in PAL an announcement $!\phi$ is assumed to have an inherent meaning: ϕ is true. This is because the semantic objects (event models) are explicitly included in the syntax as in the general DEL framework. However, the same utterance $!\phi$ (syntax) may carry different meanings (semantics) as we have seen in the we-will-see example. A closer look at public announcements should separate the utterances and their meanings. In fact, an utterance a only carries the meaning ϕ if the hearer knows that the protocol $? \phi \cdot a$ is carried out (cf. [PR03] for a detailed rationale).

To handle the protocols that carry meanings for actions, it is inevitable to introduce tests in the protocol programming language. Intuitively, the tests are not observable by the agents, unless announced previously, e.g. $[?p_{no} \cdot a_{will-see}]K_i p_{no}$ should not be valid while $[!(?p_{no} \cdot a_{will-see})][?p_{no} \cdot a_{will-see}]K_i p_{no}$ should be valid. We define the formal semantics for this enriched language in this chapter.

²Here we assume that if the protocol is announced then it is followed by all the agents. See [PS10] for an interesting discussion on “knowingly following the protocol” by agents in a setting of imperfect information.

Related work Besides the work on DEL protocols we mentioned earlier [HY09, vBGHP09, Hos09], we list some more related work here. Process logic [Pra79, HKP82] extends PDL in adding modalities to specify *progressive* behaviours like “during the execution of program π , ϕ will be true at some point.” In this chapter, we not only reason about properties in the middle of an execution of a protocol but also handle the protocol changes during the execution. Moreover, the semantics of our logics will be defined on the states in the models, instead of on paths as in [Pra79, HKP82]. Aucher [Auc09] also proposed an extended DEL, however, the reasoning of the ongoing events is facilitated, in a setting without protocols. Unlike the work of switching strategies in the context of games [PRS09], the change of our protocols can be made at any time without being planned and we also incorporate knowledge in the discussions.

Our treatment for the events that carry meaning is inspired by [PR03], in which the authors give a semantics for messages (events) according to the underlying protocol in the ETL framework. However, we can explicitly express the protocol in the language and design a semantics for the dynamics of protocols. The later feature also distinguishes our work from the work using regular expressions as protocols [BS08b, Mey87, WKvE09]. The semantics of our logics are defined on standard Kripke models, but unlike PAL, we do not use a model-changing semantics for our $[\pi]$ operator. This gives us the possibility to model radical protocol changes which are not based on the previous ones.

Structure of the chapter In this chapter we develop three logics featuring protocol changing operators, which can all be translated to PDL, but with certain convenience for modelling purposes. As an appetizer, we start in Section 4.2 with the first logic PDL[!], a version of test-free PDL equipped with protocol announcements $[\pi]$. The semantics is given in a non-standard style by using *modes* of satisfaction relations [Gab02, Wan06]. Section 4.3 extends the language PDL[!] with knowledge and Boolean tests to handle the cases like the above we-will-see example where knowing a protocol gives meanings to actions. Finally, Section 4.4 proposes a logic PDL[⊠] with automata as update models, which is powerful for modelling more complicated protocols and various interactions among agents. PDL[⊠] can be viewed as an extension of DEL [BM04, KvB04, vBvEK06] with more liberal updates.

4.2 Basic Logic PDL[!]

The formulas of PDL[!] are built from the set of basic proposition letters \mathbf{P} and the set of atomic actions Σ as follows:

$$\begin{aligned} \phi & ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid [\pi]\phi \mid [!\pi]\phi \\ \pi & ::= \mathbf{1} \mid \mathbf{0} \mid a \mid \pi \cdot \pi \mid \pi + \pi \mid \pi^* \end{aligned}$$

where $p \in \mathbf{P}$ and $a \in \Sigma$. The intended meaning of the formulas is mostly as in PDL, but “in context” of the protocol constraints: $[\pi]\phi$ now says that “after any run of the

program π which is allowed by the *current protocol*, ϕ holds". The new formula $[\pi]\phi$ expresses "after the announcement of the new protocol π , ϕ holds."

To give the semantics to PDL¹, we first recall a useful notion of regular expressions. The *input derivative* $\pi \setminus a$ of the regular expression $\pi \in \text{Reg}_\Sigma$ is defined as $\mathcal{L}(\pi \setminus a) = \{v \mid av \in \mathcal{L}(\pi)\}$. With the output function $o : \text{Reg}_\Sigma \rightarrow \{\mathbf{0}, \mathbf{1}\}$ we can axiomatize $\setminus a$ (cf. [Brz64, Con71]):

$$\begin{array}{ll} \pi = o(\pi) + \sum_{a \in \Sigma} (a \cdot \pi \setminus a) & \\ \mathbf{1} \setminus a = \mathbf{0} \setminus a = b \setminus a = \mathbf{0} \quad (a \neq b) & a \setminus a = \mathbf{1} \\ (\pi \cdot \pi') \setminus a = (\pi \setminus a) \cdot \pi' + o(\pi) \cdot (\pi' \setminus a) & (\pi + \pi') \setminus a = \pi \setminus a + \pi' \setminus a \\ (\pi)^* \setminus a = \pi \setminus a \cdot (\pi)^* & o(\pi \cdot \pi) = o(\pi) \cdot o(\pi') \\ o(\pi^*) = \mathbf{1} & o(\mathbf{1}) = \mathbf{1} \\ o(\mathbf{0}) = o(a) = \mathbf{0} & o(\pi + \pi') = o(\pi) + o(\pi') \end{array}$$

Given $w = a_0 a_1 \cdots a_n \in \Sigma^*$, let $\pi \setminus w = (\pi \setminus a_0) \setminus a_1 \cdots \setminus a_n$. It is clear that $\pi \setminus w = \{v \mid wv \in \mathcal{L}(\pi)\}$ ³. Together with the axioms of Kleene algebra [Koz91] we can syntactically derive $\pi \setminus w$ which is intuitively the *remaining protocol* of π after executing a run w . For example:

$$(a + (b \cdot c))^* \setminus b = (a \setminus b + (b \cdot c) \setminus b) \cdot (a + b \cdot c)^* = (\mathbf{0} + (\mathbf{1} \cdot c)) \cdot (a + b \cdot c)^* = c \cdot (a + (b \cdot c))^*$$

Note that in general we do not have $w \cdot (\pi \setminus w) = \pi$. We say w is *compliant with* π (notation: $w \propto \pi$) if $\pi \setminus w \neq \mathbf{0}$, namely, executing w is allowed by the protocol π .

Intuitively, to evaluate $[\pi]\phi$ we need to memorize the current protocol in some way. Here we employ a trick similar to the ones used in the semantics developed in [Gab02, Wan06, BE09]: we define the satisfaction relation w.r.t. a *mode* π (notation: \vDash_π), which is used to record the current protocol. Given the current protocol π , the allowed runs in a program π' w.r.t. π are those $w \in \Sigma^*$ such that $w \in \mathcal{L}(\pi')$ and $w \propto \pi$. Note that if the current protocol is π , then after executing a run w we have to update π by the remaining protocol $\pi \setminus w$. Now we are ready to give the semantics as follows:

$\mathcal{M}, s \vDash \phi \Leftrightarrow \mathcal{M}, s \vDash_{\Sigma^*} \phi$
$\mathcal{M}, s \vDash_\pi p \Leftrightarrow p \in V(s)$
$\mathcal{M}, s \vDash_\pi \neg \phi \Leftrightarrow \mathcal{M}, s \not\vDash_\pi \phi$
$\mathcal{M}, s \vDash_\pi \phi \wedge \psi \Leftrightarrow \mathcal{M}, s \vDash_\pi \phi \text{ and } \mathcal{M}, s \vDash_\pi \psi$
$\mathcal{M}, s \vDash_\pi [\pi']\phi \Leftrightarrow \forall (w, s') : w \in \mathcal{L}(\pi'), w \propto \pi, \text{ and } s \xrightarrow{w} s' \implies \mathcal{M}, s' \vDash_{\pi \setminus w} \phi$
$\mathcal{M}, s \vDash_\pi [!\pi']\phi \Leftrightarrow \mathcal{M}, s \vDash \langle \pi' \rangle \top \implies \mathcal{M}, s \vDash_{\pi'} \phi$

where Σ^* stands for $(a_0 + a_1 + \cdots + a_n)^*$ if $\Sigma = \{a_0, a_1, \dots, a_n\}$. The first clause says that initially everything is allowed and the last says that the newly announced protocol overrides the current one. $[\pi']\phi$ is true w.r.t. the current protocol π iff on each s' that is reachable from s by some run w of π' which is allowed by the current protocol π : ϕ holds w.r.t. the remaining protocol $\pi \setminus w$. Note that it is important to remember w which denotes *how you get to* s' as the following example shows:

³ $\pi \setminus w$ is also a regular language cf. [Con71].

4.2.1. EXAMPLE. Consider the following model \mathcal{M} :



It can be verified that:

$$\mathcal{M}, s \models [!(a \cdot c + b \cdot d)]\langle a + b \rangle (\neg \langle d \rangle \top \wedge \langle c \rangle \top \wedge [!(c + d)]\langle d \rangle \top)$$

The intuition behind this example is as follows. After announcing the protocol $a \cdot c + b \cdot d$, the program $a + b$ can be executed but actually only a can be executed on the model. Thus after executing $a + b$ only c is possible according to the remaining protocol $(a \cdot c + b \cdot d) \setminus a = c$. However, if we now announce a new protocol $(c + d)$ then d becomes available again. $\underline{\Omega}$

Recall the PDL semantics in Section 2.3.1. It is not hard to see:

4.2.2. PROPOSITION. For any test-free PDL formula ϕ and any pointed Kripke model (\mathcal{M}, s) :

$$\mathcal{M}, s \models_{\text{PDL}} \phi \iff \mathcal{M}, s \models \phi$$

A natural question to ask is whether PDL¹ is more expressive than test-free PDL. To answer the question, we now take a closer look at the strings w in the semantics of $[\pi']\phi$. Given π , let $C_{\mathcal{L}(\pi)}$ be the set of all the *pre-sequences* of π : $\{w \mid w \circ \pi\}$. We first show that we can partition $C_{\mathcal{L}(\pi)}$ into finitely many regular expressions.

4.2.3. LEMMA. For any regular expression π there is a minimal natural number k such that $C_{\mathcal{L}(\pi)}$ can be finitely partitioned into π_0, \dots, π_k and for any $w, v \in \mathcal{L}(\pi_i) : \pi \setminus w = \pi \setminus v$.

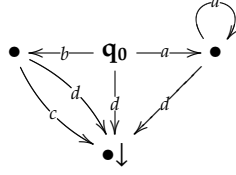
PROOF By Kleene's theorem 2.1.3 we can construct a deterministic finite automaton recognizing the language of π . It is well known that DFA can be minimized, thus we obtain a minimal automaton that recognizes $\mathcal{L}(\pi)$:

$$\mathbf{A}_\pi = (\{q_0, \dots, q_k\}, \Sigma, q_0, \succrightarrow, F)$$

where $\{q_0, \dots, q_k\}$ is a set of states with q_0 being the start state and a subset F being the set of accept states. For each $i \leq k$ such that q_i can reach a state in F : we let π_i be the regular expression corresponding to the automaton $(\{q_0, \dots, q_k\}, \Sigma, q_0, \succrightarrow, \{q_i\})$. Since \mathbf{A}_π is deterministic, it is not hard to see that these π_i form the partition that we want. \times

In the sequel, we call the above unique partition π_0, \dots, π_k the *pre-derivatives* of π . For example, the minimal deterministic automaton⁴ of $a^* \cdot d + b \cdot (c + d)$ is:

⁴We omit the transitions to the "trash" state which can not reach any accept state.



thus the pre-derivatives of $a^* \cdot d + b \cdot (c + d)$ are $\mathbf{1}, a \cdot a^*, b, a^* \cdot d + b \cdot (c + d)$ ⁵.

Now we define the following translation from PDL¹ to PDL:

$$\begin{aligned}
 t(\phi) &= t_{\Sigma^*}(\phi) \\
 t_{\pi}(p) &= p \\
 t_{\pi}(\neg\phi) &= \neg t_{\pi}(\phi) \\
 t_{\pi}(\phi_1 \wedge \phi_2) &= t_{\pi}(\phi_1) \wedge t_{\pi}(\phi_2) \\
 t_{\pi}([\pi']\phi) &= \bigwedge_{i=0}^k ([\theta_i] t_{\pi \setminus \pi_i}(\phi)) \\
 t_{\pi}(!\pi']\phi) &= \langle \pi' \rangle \top \rightarrow t_{\pi'}(\phi)
 \end{aligned}$$

where π_0, \dots, π_k are the pre-derivatives of π , θ_i is a regular expression corresponding to $\mathcal{L}(\pi') \cap \mathcal{L}(\pi_i)$, and $\pi \setminus \pi_i$ is defined as $\pi \setminus w$ for any $w \in \mathcal{L}(\pi_i)$.

By this translation we can show that PDL and PDL¹ are equally expressive.

4.2.4. THEOREM. For any pointed Kripke model \mathcal{M}, s :

$$\mathcal{M}, s \vDash \phi \iff \mathcal{M}, s \vDash_{\text{PDL}} t(\phi).$$

PROOF By induction on ϕ we can show: $\mathcal{M}, s \vDash_{\pi} \phi \iff \mathcal{M}, s \vDash_{\text{PDL}} t_{\pi}(\phi)$. The only non-trivial case is for $[\pi']\phi$:

$$\begin{aligned}
 &\mathcal{M}, s \vDash_{\pi} [\pi']\phi \\
 &\iff \forall (w, s') : w \in \mathcal{L}(\pi), w \vDash \pi', \text{ and } s \xrightarrow{w} s' \implies \mathcal{M}, s' \vDash_{\pi \setminus w} \phi \\
 &\iff \forall (w, s') : \text{if there is a pre-derivative } \pi_i : w \in \mathcal{L}(\pi), w \in \mathcal{L}(\pi_i), \text{ and } s \xrightarrow{w} s' \\
 &\text{then } \mathcal{M}, s' \vDash_{\pi \setminus w} \phi \\
 &\iff \text{for all pre-derivatives } \pi_i : \forall s' : s \xrightarrow{w} s' \text{ and } w \in \mathcal{L}(\pi), w \in \mathcal{L}(\pi_i) \text{ then } \mathcal{M}, s' \vDash_{\pi \setminus w} \phi \\
 &\iff \mathcal{M}, s \vDash \bigwedge_{i=0}^k [\theta_i] t_{\pi \setminus \pi_i}(\phi)
 \end{aligned}$$

✕

Discussion In this section, we take a rather *liberal* view on the “default” protocol, namely we assume that everything is allowed initially, and the announcements may only restrict the possible actions. On the other hand, we can well start with a *conservative* initialization where nothing is allowed unless announced later. It is not hard

⁵Note that $a \cdot a^* \cdot d + b \cdot (c + d) + d = a^* \cdot d + b \cdot (c + d)$.

to see that we can also translate this conservative version of PDL[!] to PDL if we let $t(\phi) = t_1(\phi)$ where $\mathbf{1}$ is the constant for the empty sequence i.e., the *skip* protocol. For example, $t_1([a]\perp \wedge [!a]\langle a + b \rangle \top) = [\mathbf{0}]\perp \wedge t_a(\langle a + b \rangle \top) = \langle a \rangle \top$.

Moreover, $[!\pi]$ is rather *radical* in the sense that it changes the protocol completely. We may define a more general operation as follows: Let $\pi(x) \in \text{Reg}_{\Sigma \cup \{x\}}$, namely, $\pi(x)$ is a regular expression with a variable x . Now we define:

$$\boxed{\mathcal{M}, s \vDash_{\pi} [!\pi'(x)]\phi \Leftrightarrow \mathcal{M}, s \vDash \langle \pi'(\pi) \rangle \top \implies \mathcal{M}, s \vDash_{\pi'(\pi)} \phi}$$

We can then concatenate, add, insert and repeat protocols by announcing $x \cdot \pi'$, $x + \pi'$, $\pi' + x$, and x^* respectively. It is easy to see that the announcement operator $[!\pi]$ introduced previously is a special case of $[!\pi(x)]$. We can still translate the logic with the generalized protocol announcements to PDL with an easy revision of the translation:

$$t_{\pi}([!\pi'(x)]\phi) = \langle \pi'(\pi) \rangle \top \rightarrow t_{\pi'(\pi)}(\phi)$$

4.3 Public Event Logic PDL^{!?}_b

In this section, we extend the language of PDL[!] with knowledge operator and Boolean tests in programs. We shall see that by announcing a protocol with tests, we can let actions carry propositional information as we motivated in Chapter 1. The language of PDL^{!?}_b is defined as follows:

$$\begin{aligned} \phi & ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid [\pi]\phi \mid [!\pi]\phi \mid K_i\phi \\ \pi & ::= ?\phi_b \mid a \mid \pi \cdot \pi \mid \pi + \pi \mid \pi^* \end{aligned}$$

where $i \in \mathbf{I}$ and ϕ_b are Boolean formulas based on basic propositions in \mathbf{P} . Note that we do not include $\mathbf{1}$ and $\mathbf{0}$ as atomic actions since they can be expressed by the Boolean tests $? \top$ and $? \perp$. We call the programs π in PDL^{!?}_b *guarded regular expressions*.

Now we can express the Häagen-Dazs slogan mentioned in Chapter 1 by the protocol: $\pi_{H-D} = ?p_{love} \cdot a_{buy}$. A suitable semantics should let $[!\pi_{H-D}][a_{buy}]K_i p_{love}$ be valid. However, without the announcement $!\pi_{H-D}$, the “secret” love may not be known: $[?p_{love} \cdot a_{buy}]K_i p_{love}$ should not be valid. As we mentioned in the introduction, we assume all the $a \in \Sigma$ are *public events* which can be observed by all the agents, while the tests, unless announced, are not observable to the agents.

To prepare ourselves for the definition of the semantics, we first interpret regular expressions with Boolean tests as the languages of guarded strings [Koz01]. A *guarded string* over \mathbf{P} and Σ is a sequence $\rho_1 a_1 \rho_2 a_2 \rho \dots \rho_n a_n \rho_{n+1}$ where $a_i \in \Sigma$ and $\rho_i \subseteq \mathbf{P}$ represents the valuations of basic propositions in \mathbf{P} ($p \in \rho$ iff p is true according to ρ). For any Boolean formula ψ , let $X_{\psi} \subseteq 2^{\mathbf{P}}$ be the corresponding set of valuations, represented by subsets of \mathbf{P} , that make ψ true. For any $\rho \subseteq \mathbf{P}$, let $\phi_{\rho} = \bigwedge_{p \in \rho} p \wedge \bigwedge_{p \in (\mathbf{P} - \rho)} \neg p$.

Now we can define the language of guarded strings associated with a guarded regular expression over Σ and \mathbf{P} :

$$\begin{aligned}\mathcal{L}_g(a) &= \{\rho a \rho' \mid \rho, \rho' \subseteq \mathbf{P}\} \\ \mathcal{L}_g(?\psi) &= \{\rho \mid \rho \in X_\psi\} \\ \mathcal{L}_g(\pi \cdot \pi') &= \{w \diamond v \mid w \in \mathcal{L}_g(\pi), v \in \mathcal{L}_g(\pi')\} \\ \mathcal{L}_g(\pi + \pi') &= \mathcal{L}_g(\pi) \cup \mathcal{L}_g(\pi') \\ \mathcal{L}_g(\pi^*) &= \{\epsilon\} \cup \bigcup_{n>0} (\mathcal{L}_g(\pi^n))\end{aligned}$$

where \diamond is the *fusion product*: $w \diamond v = w' \rho v'$ when $w = w' \rho$ and $v = \rho v'$; $\pi^n = \underbrace{\pi \cdots \pi}_n$.

We write $\pi_1 \equiv_g \pi_2$ if $\mathcal{L}_g(\pi_1) = \mathcal{L}_g(\pi_2)$.

4.3.1. EXAMPLE. We have:

$$?p \cdot ?q \cdot a \equiv_g ?(p \wedge q) \cdot a \equiv_g ?(p \wedge p) \cdot ?q \cdot a$$

$$?(p \wedge q) \cdot a + ?(p \wedge \neg q) \cdot a \equiv_g ?p \cdot a \text{ and } ?p \cdot a \cdot a \not\equiv_g ?p \cdot a$$

Ω

We now define the language of input derivative $\pi \setminus w$ for a guarded string w as:

$$\mathcal{L}_g(\pi \setminus w) = \{v \mid w \diamond v \in \mathcal{L}_g(\pi)\}$$

and we say $w \propto_g \pi$ if $\mathcal{L}_g(\pi \setminus w) \neq \emptyset$. As in the previous section, we let $C_\pi^g = \{w \mid w \propto_g \pi\}$.

Let $\mathcal{L}_i(w)$ be the sequence of public events $a_0 \dots a_k$ that occurs in w , e.g., $\mathcal{L}_i(?p \cdot a \cdot b) = \mathcal{L}_i(?q \cdot a \cdot ?p \cdot b) = a \cdot b$. Recall that we assume that only the public events can be observed. Thus a guarded string w is indistinguishable from another guarded string v if $\mathcal{L}_i(w) = \mathcal{L}_i(v)$.

According to the standard semantics of PAL, the effect of announcing a formula ϕ is to restrict the model to the ϕ -worlds (see Section 2.3.3). Our public events are like announcements but with preconditions given by the previously announced protocols. However, to model the public events we can also keep the model intact but remember the information induced by the public events. When evaluating epistemic formulas, we let agents only consider possible those worlds which are consistent with the previously recorded information. Since the tests are Boolean, this restriction on accessible worlds works the same as the restriction on models in standard PAL. This motivates us to use \mathbb{F}_π^ϕ in the semantics of $\text{PDL}^{!?}_b$ where ϕ is to record the information given by public events according to the protocols. We interpret $\text{PDL}^{!?}_b$ on the S5 models $(S, \mathbf{P}, \mathbf{I}, \sim_i, V)$ as follows:

$\mathcal{M}, s \vDash \phi \Leftrightarrow \mathcal{M}, s \vDash_{\Sigma^*}^{\top} \phi$
$\mathcal{M}, s \vDash_{\pi}^{\psi} p \Leftrightarrow p \in V(s)$
$\mathcal{M}, s \vDash_{\pi}^{\psi} \neg\phi \Leftrightarrow \mathcal{M}, s \not\vDash_{\pi}^{\psi} \phi$
$\mathcal{M}, s \vDash_{\pi}^{\psi} \phi \wedge \phi' \Leftrightarrow \mathcal{M}, s \vDash_{\pi}^{\psi} \phi \text{ and } \mathcal{M}, s \vDash_{\pi}^{\psi} \phi'$
$\mathcal{M}, s \vDash_{\pi}^{\psi} K_i\phi \Leftrightarrow \text{for all } v, \text{ if } s \sim_i t \text{ and } \mathcal{M}, t \vDash_{\pi}^{\psi} \psi \text{ then } \mathcal{M}, t \vDash_{\pi}^{\psi} \phi$
$\mathcal{M}, s \vDash_{\pi}^{\psi} [\pi']\phi \Leftrightarrow \forall w : w \in \mathcal{L}_g(\pi'), w \propto_g \pi, \text{ and } s[[w]]s \implies \mathcal{M}, s \vDash_{\pi \setminus w}^{\psi \wedge \phi_{\pi}^w} \phi$
$\mathcal{M}, s \vDash_{\pi}^{\psi} [!\pi']\phi \Leftrightarrow \mathcal{M}, s \vDash \langle \pi' \rangle \top \implies \mathcal{M}, s \vDash_{\pi'}^{\psi} \phi$

where:

$$s[[w]]s \iff w = \rho a_1 \rho a_2 \rho \cdots \rho a_k \rho \text{ and } V(s) = \rho$$

and

$$\phi_{\pi}^w = \bigvee \{ \phi_{\rho} \mid v = \rho a_1 \rho a_2 \rho \cdots \rho a_k \rho, \mathcal{L}_i(w) = \mathcal{L}_i(v), v \propto_g \pi \}$$

Note that we do not include the transitions labelled by $a \in \Sigma$ in the models since we assume that each public event is executable at each state unless it is not compliant with the current protocol (e.g., you can talk about anything in public unless constrained by some law or conventions). Since the public events are intended to be announcement-like events, we also assume that executing a protocol of such event does not result in changing the real state from one to another. This explains the uniformity of ρ and s in the definition of $[[w]]$. Now we explain the ideas behind ϕ_{π}^w as follows. First given a $w = \rho a_1 \rho a_2 \rho \cdots \rho a_k \rho$, we collect all the sequences $v = \rho' a_1 \rho' a_2 \rho' \cdots \rho' a_k \rho'$ such that $v \propto_g \pi$. Intuitively ρ' represents the information carried by v according to the protocol π . Since each such v is indistinguishable from w for all the agents, the disjunction ϕ_{π}^w is then the information which can be derived from the observation of the public events in w according to the protocol π .

Consider the Häagen-Dazs example, let \mathcal{M} be a two-world model representing that a girl i does not know whether a boy loves her or not (she is not sure between a p_{love} -world s and a $\neg p_{love}$ world t). Let $\pi = ?p_{love} \cdot a_{buy}$, and $w_0 = \{p_{love}\} a_{buy} \{p_{love}\}$. It is clear that w_0 is the only guarded string in $\mathcal{L}_g(\pi)$ that have an uniform ρ . Note that $\mathcal{L}_i(w_0) = \mathcal{L}_i(\emptyset a_{buy} \emptyset)$, thus $\phi_{\Sigma^*}^{w_0} = p \vee \neg p$. We now show $\mathcal{M}, s \not\vDash [\pi] K_i p_{love}$:

$$\begin{aligned} & \mathcal{M}, s \vDash [\pi] K_i p_{love} \\ \iff & \mathcal{M}, s \vDash_{\Sigma^*}^{\top} [\pi] K_i p_{love} \\ \iff & \text{for all } w \in \mathcal{L}_g(\pi), w \propto_g \Sigma^*, \text{ and } s[[w]]s \implies \mathcal{M}, s \vDash_{\Sigma^* \setminus w}^{\phi_{\Sigma^*}^w} K_i p_{love} \\ \iff & s[[w_0]]s \implies \mathcal{M}, s \vDash_{\Sigma^* \setminus w_0}^{\phi_{\Sigma^*}^{w_0}} K_i p_{love} \\ \iff & \mathcal{M}, s \vDash_{?p \cdot \Sigma^*}^{\phi_{\Sigma^*}^{w_0}} K_i p_{love} \\ \iff & \mathcal{M}, s \vDash_{?p \cdot \Sigma^*}^{p \vee \neg p} K_i p_{love} \end{aligned}$$

Since $s \sim_i t$ and $\mathcal{M}, t \vDash p \vee \neg p$ then $\mathcal{M}, s \not\vDash [\pi] K_i p_{love}$. On the other hand:

$$\begin{aligned}
& \mathcal{M}, s \models [!\pi][\pi]K_i p_{love} \\
\iff & \mathcal{M}, s \models_{\pi}^{\top} [\pi]K_i p_{love} \\
\iff & \mathcal{M}, s \models_{\pi \setminus w_0}^{\phi_{\pi}^{w_0}} K_i p_{love} \\
\iff & \mathcal{M}, s \models_{\top}^{\beta} K_i p_{love}
\end{aligned}$$

Therefore $\mathcal{M}, s \models [!\pi][\pi]K_i p_{love}$.

Similarly, for the we-will-see scenario mentioned in the introduction, if \mathcal{M} is a two-world model representing that a Westerner i does know whether p_{no} (state s) or $\neg p_{no}$ (state t) then we can show that:

$$\mathcal{M}, s \models [!(\top \cdot a_{will-see})][(\top \cdot a_{will-see})\neg K_i p_{no} \wedge [!(\top \cdot a_{will-see})][(\top \cdot a_{will-see})K_i p_{no}]$$

where $\top \cdot a_{will-see}$ is the default protocol a Westerner may have as the standard interpretation for the sentence “we will see” which does not carry any useful information.

In the rest of this section we will show that $\text{PDL}^{!b}$ can be translated back to PDL as well. We will follow a similar strategy as in the previous section to finitely partition C_{π}^g . This time we need to use automata on guarded strings. Given \mathbf{P} let $\mathcal{B}(\mathbf{P})$ be the set $2^{2^{\mathbf{P}}}$. Intuitively, $X \in \mathcal{B}(\mathbf{P})$ represent Boolean formulas over \mathbf{P} .

4.3.2. DEFINITION. (Automata on guarded strings [Koz01]) A finite automaton on guarded strings (or simply guarded automaton) over a finite set of actions Σ and a finite set of atomic tests \mathbf{P} is a tuple $\mathbf{A} = (Q, \Sigma, \mathbf{P}, q_0, \rightarrow, F)$ where the transitions are labelled by atomic actions in Σ (action transitions) and sets $X \in \mathcal{B}(\mathbf{P})$ (test transitions). \mathbf{A} accepts a finite string w over $\Sigma \cup \mathcal{B}(\mathbf{P})$ (notation: $w \in \mathcal{L}_{\Sigma \cup \mathcal{B}(\mathbf{P})}(\mathbf{A})$), if it accepts w as a standard finite automaton over label set $\Sigma \cup \mathcal{B}(\mathbf{P})$. The acceptance for guarded strings is defined based on the acceptance of normal strings and the following transformation function G which takes a string over $\Sigma \cup \mathcal{B}(\mathbf{P})$ and outputs a set of guarded strings.

$$\begin{aligned}
G(a) &= \{\rho a \rho' \mid \rho, \rho' \subseteq \mathbf{P}\} \\
G(X) &= \{\rho \mid \rho \in X\} \\
G(w w') &= \{v \rho v' \mid v \rho \in G(w) \text{ and } \rho v' \in G(w')\}
\end{aligned}$$

We say \mathbf{A} accepts a finite guarded string $v : \rho_0 a_0 \rho_1 \dots a_{k-1} \rho_k$ over Σ and \mathbf{P} , if $v \in G(w)$ for some string $w \in \mathcal{L}_{\Sigma \cup \mathcal{B}(\mathbf{P})}(\mathbf{A})$. Let $\mathcal{L}_g(\mathbf{A})$ be the language of guarded strings accepted by \mathbf{A} . □

We say a guarded automaton is *deterministic* if the following hold (cf. [Koz01]):

- Each state is either a state that only has outgoing action transitions (*action state*) or a state that only has outgoing test transitions (*test state*).
- The outgoing action transitions are deterministic: for each action state q and each $a \in \Sigma$, q has one and only one a -successor.
- The outgoing test transitions are deterministic: they are labelled by $\{\{\rho\} \mid \rho \subseteq \mathbf{P}\}$ and for each test state q and each ρ , q has one and only one $\{\rho\}$ -successor. Clearly these tests ρ at a test state are logically pairwise exclusive and altogether exhaustive (viewing ρ as the Boolean formula ϕ_{ρ}).

- The start state is a test state and all accept states are action states.
- Each cycle contains at least one action transition.

The Kleene theorem between guarded automata and guarded regular expressions is proved in [Koz01].

4.3.3. THEOREM. [Koz01, Theorem 3.1, 3.4] *For each guarded regular expression π over \mathbf{P} and Σ there is a (deterministic) guarded automaton \mathbf{A} over \mathbf{P} and Σ such that $\mathcal{L}_g(\pi) = \mathcal{L}_g(\mathbf{A})$, and vice versa.*

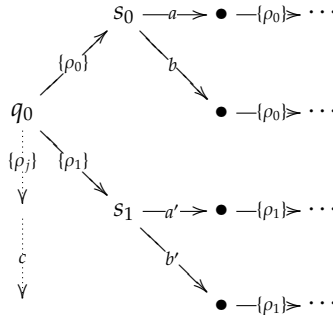
Let L_U be the language of ρ -uniform guarded strings: $\{\rho a_1 \rho a_2 \rho \cdots \rho a_k \rho \mid \rho \subseteq \mathbf{P}, a_i \in \Sigma\}$. Clearly there is a regular expression for this language: $\sum_{\rho \subseteq \mathbf{P}} ((\phi_\rho \cdot (a_1 + \cdots + a_m))^* \cdot \phi_\rho)$ if $\Sigma = \{a_1, \dots, a_m\}$. Let $U_\pi^g = C_\pi^g \cap L_U$ be the ρ -uniform part of C_π^g . Following the idea in the previous section, we first need to prove the following lemma:

4.3.4. LEMMA. *Given a guarded regular expression π over Σ and \mathbf{P} , we can finitely partition U_π^g into π_0, \dots, π_n such that for any $i \leq k : w, v \in \mathcal{L}_g(\pi_i) \implies \pi_i \setminus w = \pi_i \setminus v$ and $\phi_\pi^w = \phi_\pi^v$.*

PROOF (Sketch) The strategy for the proof is as follows: we first partition U_π^g into π_0, \dots, π_n such that for any $i \leq n$, for any $w, v \in \mathcal{L}_g(\pi_i) : \phi_\pi^w = \phi_\pi^v$, then we further partition each π_i according to the shared derivatives like in Lemma 4.2.3.

From Theorem 4.3.3, we can build deterministic guarded automata \mathbf{A}_π and \mathbf{A}_U such that $\mathcal{L}_g(\mathbf{A}_\pi) = \mathcal{L}_g(\pi)$ and $\mathcal{L}_g(\mathbf{A}_U) = L_U$. From the definition of deterministic guarded automata, we can assume that in such deterministic automata test states can only have action states as successors, for otherwise the successor test states can be pruned⁶. Now set all the action states in \mathbf{A}_π that can reach some accept states as the new accept states, we can obtain a guarded automaton \mathbf{A}_C such that $\mathcal{L}_g(\mathbf{A}_C) = C_\pi^g$. Finally we can build an automaton \mathbf{A} such that $\mathcal{L}_g(\mathbf{A}) = C_\pi^g \cap L_0$ by the usual automata product of \mathbf{A}_C and \mathbf{A}_U .

It is not hard to see that if you start with a ρ transition in \mathbf{A} then you can never go through a ρ' transition which leads to an accept state such that $\rho \neq \rho'$. Thus the automaton is in the following shape if all the states that are not leading to any accept states are pruned:



⁶Since all the test transitions are labelled by $\{\rho\}$ for some $\rho \subseteq \mathbf{P}$, two consecutive tests are either identical or logically exclusive.

We can then separate ρ_i “zones” from each other by taking each s_i as the start state for zone ρ_i . Let \mathbf{B}_{s_i} be the standard finite automata over action set $\Sigma : (Q_{act}, \Sigma, s_i, \succrightarrow, F)$ where Q_{act} is the set of action states in Q , F is the set of accept states of \mathbf{A} that are also in Q_{act} , and $q \xrightarrow{a} q' \iff q \xrightarrow{a} \overset{\rho_i}{\rightarrow} q'$ in \mathbf{A} . Given $Z \subseteq \{\rho_0, \dots, \rho_k\}$ (intuitively a Boolean formula), let \mathbf{D}_Z be the product automaton $\prod_{\rho_i \in Z} \mathbf{B}_{s_i} \times \prod_{\rho_i \notin Z} \overline{\mathbf{B}_{s_i}}$ where $\overline{\mathbf{B}_{s_i}}$ is the complement automaton of \mathbf{B}_{s_i} . We can show that \mathbf{D}_Z recognizes all the sequences $w \in \Sigma^*$ such that $\{\rho \mid w = \mathcal{L}_i(v) \text{ for some } v = \rho a_1 \rho \dots \rho a_k \rho \in \mathcal{L}_g(\mathbf{A})\} = Z$. Then without much effort, we can turn \mathbf{D}_Z into a finite guarded automaton which recognizes guarded strings v in C_π^g , such that:

$$\phi_\pi^v = \bigvee \{ \phi_\rho \mid v' = \rho a_1 \rho a_2 \rho \dots \rho a_k \rho, \mathcal{L}_i(v) = \mathcal{L}_i(v'), v' \alpha_g \pi \} = \phi_Z$$

Thus C_π^g can be partitioned into finitely many regular expressions π_i such that for any $w, v \in \mathcal{L}_g(\pi_i) : \phi_\pi^w = \phi_\pi^v$. By the similar techniques as in the previous section, we can further partition each of these regular expressions π_i into finitely many regular expressions $\pi_{i0} \dots \pi_{im}$ such that for any $w, v \in \pi_{ij} : \pi \setminus w = \pi \setminus v$. Thus we can partition C_π^g into $\pi_{00}, \dots, \pi_{km}$ w.r.t $\phi_{00}, \dots, \phi_{km}$ and $\pi'_{00}, \dots, \pi'_{km}$ such that for any $i \leq k, j \leq m : w, v \in \mathcal{L}_g(\pi_{ij}) \implies \pi \setminus w = \pi \setminus v = \pi''_{ij}$ and $\phi_\pi^w = \phi_\pi^v = \phi_{ij}$. \star

Now we define the following translation from PDL^{12b} to its fragment without $[!\pi]$ which is a $\text{PDL}_{\Sigma \cup \text{UI}}$ language with Boolean tests:

$$\begin{aligned} t(\phi) &= t_{\Sigma^*}^\top(\phi) \\ t_\pi^\psi(p) &= p \\ t_\pi^\psi(\neg\phi) &= \neg t_\pi^\psi(\phi) \\ t_\pi^\psi(\phi_1 \wedge \phi_2) &= t_\pi^\psi(\phi_1) \wedge t_\pi^\psi(\phi_2) \\ t_\pi^\psi(K_i\phi) &= [i](t_\pi^\psi(\psi) \rightarrow t_\pi^\psi(\phi)) \\ t_\pi^\psi([\pi']\phi) &= \bigwedge_{i=0}^n ([\theta_i] t_{\pi \setminus \pi_i}^{\psi \wedge \phi_i}(\phi)) \\ t_\pi^\psi([\! \pi']\phi) &= \langle \pi' \rangle \top \rightarrow t_{\pi'}^\psi(\phi) \end{aligned}$$

where π_0, \dots, π_k form a partition of U_π^g satisfying the requirements stated by the above lemma. θ_i is a regular expression corresponding to $\mathcal{L}_g(\pi') \cap \mathcal{L}_g(\pi_i)$, and $\pi \setminus \pi_i$ is $\pi \setminus w$ for any $w \in \mathcal{L}_g(\pi_i)$ and ϕ_i is ϕ_π^w for any $w \in \mathcal{L}_g(\pi_i)$.⁷

Note that the translated formulas still have two kinds of modalities: $[\pi]$ and $[i]$. We now argue that we can further eliminate the program modalities. Since we assumed that every event $a \in \Sigma$ is executable at any state when we disregard the protocol constraint, then we can actually replace each $a \in \Sigma$ with $? \top$ in the translated formula. Now the program modalities appearing in the translated formula are action-free. Without much effort, we can convert a regular expression of tests into a single test e.g., $(? \phi + ? \psi)^* ? \chi$ is equivalent to $?(((\phi \vee \psi) \vee \top) \wedge \chi)$. Finally we can eliminate such

⁷Note that if ψ is Boolean then $t_\pi^\psi(\psi) = \psi$, e.g., in $t_\pi^\psi(K_i\phi)$.

test modalities by the validity: $[?\phi]\psi \leftrightarrow (\phi \rightarrow \psi)$. Let $t'(\phi)$ be the formula which is obtained from $t(\phi)$ by further eliminating program modalities as described above, then we can translate $\text{PDL}^{!b}$ to basic epistemic logic (EL_1) (thus also to PDL_1).

4.3.5. THEOREM. *For any pointed S5 Kripke model $\mathcal{M} = (S, \mathbf{P}, \mathbf{I}, \sim_i, V, s)$:*

$$\mathcal{M}, s \models_{\text{PDL}^{!b}} \phi \iff \mathcal{M}, s \models_{\text{EL}} t'(\phi).$$

As an example, consider the formula $\phi = [!(?p \cdot a) + b][a]K_i p$:

$$\begin{aligned} t(\phi) &= t_{\Sigma}^{\top}(\phi) \\ &= \langle (?p \cdot a) + b \rangle \top \rightarrow t_{((?p \cdot a) + b)}^{\top}([a]K_i p) \\ &= \langle (?p \cdot a) + b \rangle \top \rightarrow [?p \cdot a]t_{?p}^p(K_i p) \\ &= \langle (?p \cdot a) + b \rangle \top \rightarrow [?p \cdot a][i](p \rightarrow p) \end{aligned}$$

Therefore by replacing a and b with $? \top$ we have:

$$\langle (?p \cdot ? \top) + ? \top \rangle \top \rightarrow ([?p \cdot ? \top][i](p \rightarrow p))$$

It is easy to see that the above formula is logically equivalent to $\langle ?p \rangle \top \rightarrow ([?p][i]\top)$ which is equivalent to \top . Indeed, $[!(?p \cdot a) + b][a]K_i p$ is a valid formula.

4.4 Update Logic PDL[⊠]

$\text{PDL}^!$ and $\text{PDL}^{!b}$ presented in the previous sections are limited in their convenience for modelling epistemic protocols due to the following issues:

- The restriction to Boolean tests excludes the possibility of handling protocols with more complicated pre-conditions, e.g., $?K_i p \cdot a$: *if you know p then do a .*
- The protocols are interpreted as languages of strings, thus we cannot handle branching structures which are useful when considering branching protocols e.g., strategies in games.
- $\text{PDL}^{!b}$ does not allow complicated epistemic actions as in DEL [BM04], but only public events.
- The changes of protocols are assumed to be public and agents do not have initial uncertainties about protocols.

As we have seen in the previous sections, operations on finite automata are crucial in proving various results. A natural idea is to use automata directly as modalities in the language. Inspired by [KvB04], in this section we generalize the notion of event models in DEL and introduce a version of PDL with product modalities taking automata as arguments such that the above issues can be handled.

To encode initial uncertainties of protocols we first need to enrich Kripke models with protocol information. Following the notion in process algebra, we use Kripke models with (successful) termination:

4.4.1. DEFINITION. (Kripke Model with Termination) A Kripke model with termination (KMT) is a tuple $\mathcal{M} = (S, \mathbf{P}, \Sigma, \rightarrow, V, F)$ where $(S, \mathbf{P}, \Sigma, \rightarrow, V)$ is a standard Kripke model and $F \subseteq S$ is a set of terminating states. We also write $s \downarrow$ for $s \in F$. A pointed KMT is a KMT with a designated state in it. \mathfrak{M}

Intuitively, the protocol encoded at a state in a KMT can be “read off” by viewing the KMT as an automaton with the designated state as the start state and terminating states as the accept states. A classic Kripke model $\mathcal{M} = (S, \mathbf{P}, \Sigma, \rightarrow, V)$ can be viewed as a KMT with the universal termination: $\mathcal{M}^\downarrow = (S, \mathbf{P}, \Sigma, \rightarrow, V, S)$. The uncertainties of the initial protocols can be modelled by epistemic relations among those states where different protocols are encoded. Bisimulation on KMTs can be defined in a straightforward way:

4.4.2. DEFINITION. (Bisimulation on KMT (\Leftrightarrow)) A binary relation R between the domain of two KMTs $(S, \mathbf{P}, \Sigma, \rightarrow, V, F)$ and $(S', \mathbf{P}', \Sigma', \rightarrow', V', F')$ is called bisimulation iff $(s, s') \in R$ implies that the following conditions hold:

- $s \in F \iff s' \in F'$;
- $p \in V(s) \iff p \in V(s')$;
- if $s \xrightarrow{a} t$ then there exist t' such that $s' \xrightarrow{a} t'$ and tRt' ;
- if $s' \xrightarrow{a} t'$ then there exist t such that $s \xrightarrow{a} t$ and tRt' .

\mathfrak{M}

In this section we build our PDL-style language by using finite automata in two ways: first as program modalities which are alternative representations of modalities with regular expressions with tests as in PDL (cf., e.g., [HKT00]); and second, as update models, the generalized counterpart of the protocol announcements in PDL¹ and PDL^{1?b}. The formulas of our update logic PDL[⊠] are built from \mathbf{P} and Σ as follows:

$$\phi ::= \top \mid \downarrow \mid p \mid \neg\phi \mid \phi \wedge \phi \mid [\mathbf{A}]\phi \mid [\boxtimes\mathbf{A}]\phi$$

where $p \in \mathbf{P}$, \downarrow is a constant for successful termination, and each $\mathbf{A} = (Q, \Phi, \Sigma, \rightarrow, G, q_0)$ is an automaton over actions in Σ and tests in a finite set Φ of PDL[⊠] formulas⁸. Intuitively, $[\mathbf{A}]\phi$ says “after any execution of the program encoded by \mathbf{A} , ϕ holds”. $[\boxtimes\mathbf{A}]\phi$ expresses “after updating the current protocol with the one encoded by \mathbf{A} , ϕ holds”. To simplify the notation, we sometimes use $[\pi^{\mathbf{A}}]$ to denote the automaton modality corresponding to the regular expression with test π .

The semantics for the crucial formulas is given as follows⁹:

$\mathcal{M}, s \vDash \downarrow$	\iff	$s \in F$
$\mathcal{M}, s \vDash [\mathbf{A}]\phi$	\iff	for all $s' : s \ll [w]s'$ and $w \in \mathcal{L}(\mathbf{A}) \implies \mathcal{M}, s' \vDash \phi$
$\mathcal{M}, s \vDash [\boxtimes\mathbf{A}]\phi$	\iff	$(\mathcal{M}, s) \boxtimes \mathbf{A} \vDash \phi$

⁸The formulas in Φ should be constructed at the earlier stages of the mutual induction on PDL[⊠] formulas ϕ and automata \mathbf{A} .

⁹According to the semantics, $\boxtimes\mathbf{A}$ is an unconditional update, thus $[\boxtimes\mathbf{A}]\phi \leftrightarrow \langle \boxtimes\mathbf{A} \rangle \phi$ is valid.

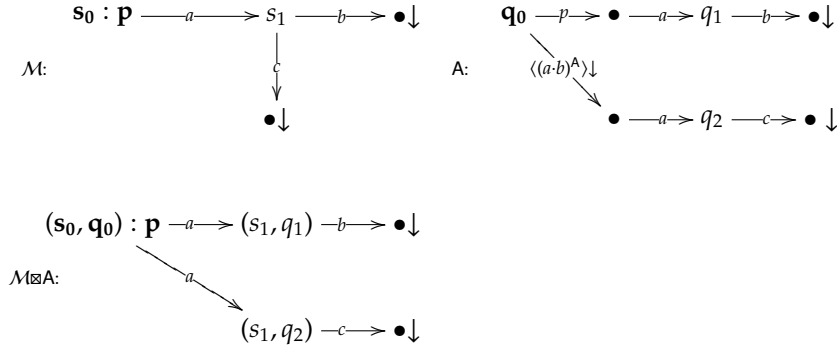
where $s[[w]]s$ is defined as on page 14 for the standard PDL, and the operation \boxtimes is defined as:

4.4.3. DEFINITION. (Update Product \boxtimes) Given a KMT $\mathcal{M} = (S, \mathbf{P}, \Sigma, \rightarrow, V, F)$ and a guarded automaton $\mathbf{A} = (Q, \Phi, \Sigma, q_0, \succrightarrow, G)$, the product model is a KMT: $(\mathcal{M} \boxtimes \mathbf{A}) = (S', \mathbf{P}, \Sigma, \rightarrow', V', F')$ where:

$$\begin{aligned} S' &= S \times Q \\ \xrightarrow{a'} &= \{(s, q), (s', q') \mid s \xrightarrow{a} s', q \xrightarrow{\bar{\phi}} \xrightarrow{a} q', \text{ and } \mathcal{M}, s \models \bigwedge \bar{\phi}\} \\ F' &= \{(s, q) \mid s \in F, \exists q' \in G : q \xrightarrow{\bar{\phi}} q' \text{ and } \mathcal{M}, s \models \bigwedge \bar{\phi}\} \\ V'((s, q)) &= V(s) \end{aligned}$$

where $\bar{\phi}$ is a possibly empty sequence of tests in Φ . We let $\bigwedge \bar{\phi}$ be the conjunction of the formulas in $\bar{\phi}$ and let it be \top if $\bar{\phi}$ is empty. For pointed models: $(\mathcal{M}, s_0) \boxtimes \mathbf{A}$ is defined as $\mathcal{M} \boxtimes \mathbf{A}, (s_0, q_0)$. \mathfrak{M}

4.4.4. EXAMPLE. We only name a few important states e.g., s_0 in \mathcal{M} where p holds. In the product model $\mathcal{M} \boxtimes \mathbf{A}$ below, we only show the generated submodel w.r.t. (s_0, q_0) :

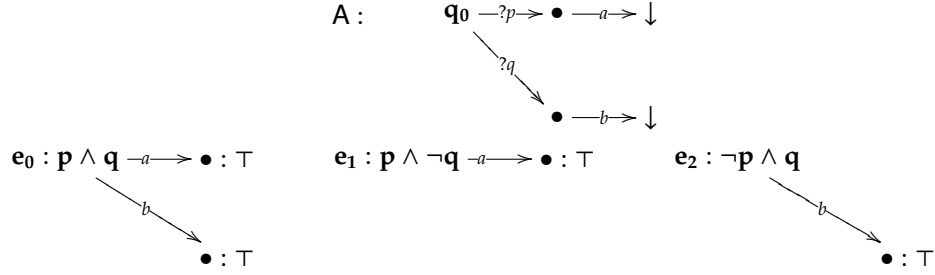


In \mathcal{M} , after executing a we can have a choice of b and c (both can lead to the successful termination). \mathbf{A} encodes the protocol: *if p then do $a \cdot b$ and if $a \cdot b$ is possible then do $a \cdot c$* . Updating \mathbf{A} on \mathcal{M} we obtain $\mathcal{M} \boxtimes \mathbf{A}$ where the choice of b and c after executing a is no longer possible, instead we need to make the choice of $a \cdot b$ and $a \cdot c$ at the beginning. According to the semantics: $\mathcal{M}, s_0 \models [a^A](\langle b^A \rangle \downarrow \wedge \langle c^A \rangle \downarrow) \wedge [\boxtimes \mathbf{A}][a^A](\neg(\langle b^A \rangle \downarrow \wedge \langle c^A \rangle \downarrow)) \Omega$

As observed in [KvB04], we can view a classic event model of [BMS98] as a automaton where each state with outgoing transitions is guarded by a unique test (the precondition of the state in the event model).

On the other hand, our guarded automata based updates give us more freedom in modelling protocols compared to the event models. Consider the following simple update model denoting the protocol “*if p then you do a and if q then you do b* ”. It cannot be mimicked by any single-pointed event model. Instead the update can be simulated by a multi-pointed event model combining three single-pointed event models

with mutually exclusive preconditions at the designated worlds (if we disregard the termination information):



where $(e : \phi)$ denotes that the precondition of e is ϕ .

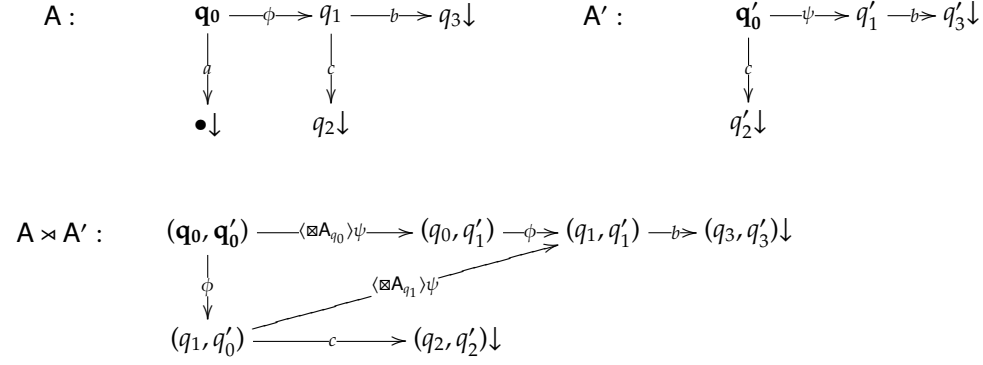
In the sequel, given an automaton A , we let A_q be the automaton as A but with *start state* q and let A^q be the automaton as A but with q as the only *accept state*.

[KvB04] shows that PDL with event model update is equally as expressive as PDL itself by defining a translation pushing the product operators to the inner part of the formulas in order to eliminate them in the end. We will show PDL^\boxtimes can be translated back to PDL as well by following the same idea. In particular, for the formula in the shape of $[\boxtimes A][B]\phi$, we need to translate it into some formula in the shape of $[A'][\boxtimes B']\psi$. Namely we need to mimic the program B after the update $\boxtimes A$ by some program before the update. For this purpose, we first define a new product between automata A and A' to handle the interaction between modalities $[A]$ and $[\boxtimes A]$.

4.4.5. DEFINITION. (Sequential Product \times) Given two automata $A = (Q, \Phi, \Sigma, \succ, q_0, F)$ and $A' = (Q', \Phi', \Sigma, \succ, q'_0, F')$ the sequential product $A \times A'$ is again an automaton: $(Q^\times, \Phi \cup \Phi'', \Sigma, \succ^\times, (q_0, q'_0), F^\times)$ where:

$$\begin{aligned}
 \Phi'' &= \{ \langle A_q \rangle \psi \mid \psi \in \Phi', q \in Q \} \\
 Q^\times &= Q \times Q' \\
 \succ^{\times a} &= \{ ((q_1, q'_1), (q_2, q'_2)) \mid q_1 \xrightarrow{a} q_2 \text{ and } q'_1 \xrightarrow{a} q'_2 \} \\
 \succ^{\times \phi} &= \{ ((q_1, q'_1), (q_2, q'_2)) \mid (\phi = \langle \boxtimes A_{q_1} \rangle \psi, q_1 = q_2 \text{ and } q'_1 \xrightarrow{\psi} q'_2) \\
 &\quad \text{or } (q_1 \xrightarrow{\phi} q_2 \text{ and } q'_1 = q'_2) \} \\
 F^\times &= \{ (q, q') \mid q' \in F' \}
 \end{aligned}$$

Here is an example of a sequential product:



Let PDL^\downarrow be the $[\boxtimes \mathbf{A}]$ -free fragment of PDL^\boxtimes . We can then define a translation t from the language of PDL^\boxtimes to the language of PDL^\downarrow by pushing $\boxtimes \mathbf{A}$ through other modalities (cf. [KvB04]):

$$\begin{aligned}
 t(\top) &= \top \\
 t(\downarrow) &= \downarrow \\
 t(p) &= p \\
 t(\neg\phi) &= \neg t(\phi) \\
 t(\phi_1 \wedge \phi_2) &= t(\phi_1) \wedge t(\phi_2) \\
 t([\mathbf{A}]\phi) &= [t(\mathbf{A})]t(\phi) \\
 t([\boxtimes \mathbf{A}]\top) &= \top \\
 t([\boxtimes \mathbf{A}]\downarrow) &= \downarrow \wedge t(\chi_{\mathbf{A}}) \\
 t([\boxtimes \mathbf{A}]p) &= p \\
 t([\boxtimes \mathbf{A}]\neg\phi) &= \neg t([\boxtimes \mathbf{A}]\phi) \\
 t([\boxtimes \mathbf{A}](\phi_1 \wedge \phi_2)) &= t([\boxtimes \mathbf{A}]\phi_1) \wedge t([\boxtimes \mathbf{A}]\phi_2) \\
 t([\boxtimes \mathbf{A}][\mathbf{B}]\phi) &= \bigwedge_{q \in Q} t([\mathbf{A}^q \times \mathbf{B}][\boxtimes \mathbf{A}_q]\phi) \\
 t([\boxtimes \mathbf{A}][\boxtimes \mathbf{B}]\phi) &= t([\boxtimes \mathbf{A}]t([\boxtimes \mathbf{B}]\phi))
 \end{aligned}$$

where $t(\mathbf{A})$ is the automaton where each test label ψ in \mathbf{A} is replaced by $t(\psi)$, and $\chi_{\mathbf{A}} = \bigvee \{ \bigwedge \bar{\phi} \mid \bar{\phi} \in \mathcal{L}(\mathbf{A}) \}$. Intuitively, $\chi_{\mathbf{A}}$ is the “termination test” in \mathbf{A} : the disjunction of the combined tests which can lead to an accept state without going through any action transitions. Note that $\chi_{\mathbf{A}}$ can not be an infinite disjunction essentially since we assume the set of test labels is finite and thus modulo logical equivalence there are only finitely many $\bigwedge \bar{\phi}$. $\chi_{\mathbf{A}}$ can be computed as follows: we revise \mathbf{A} by only keeping the accept states that are reachable from the start state in \mathbf{A} via test transitions only; then we can turn this new finite automaton into a regular expression of tests; finally we turn this regular expression into a formula as we mentioned in the end of the previous section. By proving the faithfulness of the translation we can show:

4.4.6. THEOREM. PDL^\boxtimes over KMT is equally expressive as PDL^\downarrow over KMT.

PROOF (sketch) The non-trivial case is to check that $\langle \boxtimes \mathbf{A} \rangle \langle \mathbf{B} \rangle \phi \leftrightarrow \bigvee_{q \in Q} \langle \mathbf{A}^q \times \mathbf{B} \rangle \langle \boxtimes \mathbf{A}_q \rangle \phi$ is valid.

\Rightarrow : Suppose $\mathcal{M}, s_0 \models \langle \boxtimes \mathbf{A} \rangle \langle \mathbf{B} \rangle \phi$ then $(\mathcal{M}, s_0) \boxtimes \mathbf{A} \models \langle \mathbf{B} \rangle \phi$. Thus there is a path $v \in \mathcal{L}(\mathbf{B})$ such that $(s_0, q_0) \llbracket v \rrbracket (s, q)$ for some (s, q) in $(\mathcal{M}, s_0) \boxtimes \mathbf{A}$ and $\mathcal{M} \boxtimes \mathbf{A}, (s, q) \models \phi$ (equivalently $\mathcal{M}, s \models \langle \boxtimes \mathbf{A}_q \rangle \phi$). In order to show $\mathcal{M}, s_0 \models \langle \mathbf{A}^q \times \mathbf{B} \rangle \langle \boxtimes \mathbf{A}_q \rangle \phi$, we need to find a $w \in \mathcal{L}(\mathbf{A}^q \times \mathbf{B})$ such that $s_0 \llbracket w \rrbracket s$. The spirit of the proof is to match sequences as follows:

$$\begin{array}{l}
\text{positions:} \quad 0 \dots \quad k \quad \quad k+1 \quad \quad \quad k+2 \quad \quad \dots n \\
\\
\exists v : \quad \dots \xrightarrow{\psi} \xrightarrow{a} \dots \\
\\
\text{in } \mathbf{B} : \quad q'_0 \dots q'_k \xrightarrow{\psi} q'_{k+1} \xrightarrow{a} q'_{k+2} \dots q'_n \downarrow \\
\\
\text{in } \mathcal{M} \boxtimes \mathbf{A} : \quad (s_0, q_0) \dots (s_k, q_k) \xrightarrow{\psi} (s_k, q_k) \xrightarrow{a} (s_{k+2}, q_{k+2}) \dots (s, q) \models \phi \\
\\
\text{in } \mathbf{A}^q : \quad q_0 \dots q_k \xrightarrow{\bar{\psi}'} q_{k+1} \xrightarrow{a} q_{k+2} \dots q \downarrow \\
\\
\text{let } w \text{ be :} \quad \dots \xrightarrow{\langle \boxtimes \mathbf{A}_{q_k} \rangle \psi} \xrightarrow{\bar{\psi}'} \xrightarrow{a} \dots \\
\\
\text{in } \mathbf{A}^q \times \mathbf{B} : \quad (q_0, q'_0) \dots (q_k, q'_k) \xrightarrow{\langle \boxtimes \mathbf{A}_{q_k} \rangle \psi} (q_k, q'_{k+1}) \xrightarrow{\bar{\psi}'} (q_{k+1}, q'_{k+1}) \xrightarrow{a} (q_{k+2}, q'_{k+2}) \dots (q, q'_n) \downarrow \\
\\
\text{in } \mathcal{M} : \quad s_0 \dots s_k \xrightarrow{\langle \boxtimes \mathbf{A}_{q_k} \rangle \psi} s_k \xrightarrow{\wedge \bar{\psi}'} s_k \xrightarrow{a} s_{k+2} \dots s \models \langle \boxtimes \mathbf{A}_q \rangle \phi
\end{array}$$

where $\bar{\psi}'$ represents a sequence of transitions labelled by the sequence $\bar{\psi}'$.

\Leftarrow : Suppose there is a q in \mathbf{A} such that $\mathcal{M}, s_0 \models \langle \mathbf{A}^q \times \mathbf{B} \rangle \langle \boxtimes \mathbf{A}_q \rangle \phi$. Then there is a $w \in \mathcal{L}(\mathbf{A}^q \times \mathbf{B})$ such that there is an s in $\mathcal{M} : s_0 \llbracket w \rrbracket s$ and $\mathcal{M}, s \models \langle \boxtimes \mathbf{A}_q \rangle \phi$ (equivalently $\mathcal{M} \boxtimes \mathbf{A}, (s, q) \models \phi$). To prove $\langle \boxtimes \mathbf{A} \rangle \langle \mathbf{B} \rangle \phi$, we only need to find some $v \in \mathcal{L}(\mathbf{B})$ such that $(s_0, q_0) \llbracket v \rrbracket (s, q)$ in $\mathcal{M} \boxtimes \mathbf{A}$. We demonstrate the idea of the proof as follows:

$$\begin{array}{l}
\text{positions:} \quad 0 \dots \quad k \quad \quad k+1 \quad \quad k+2 \quad \quad k+3 \quad \quad \dots n \\
\\
\exists w : \quad \dots \xrightarrow{\langle \boxtimes A_{q_k} \rangle \psi} \xrightarrow{\psi'} \xrightarrow{a} \dots \\
\\
\text{in } A^q \times B : \quad (q_0, q'_0) \dots (q_k, q'_k) \xrightarrow{\langle \boxtimes A_{q_k} \rangle \psi} (q_k, q'_{k+1}) \xrightarrow{\psi'} (q_{k+2}, q'_{k+1}) \xrightarrow{a} (q_{k+3}, q'_{k+3}) \dots (q, q'_n) \downarrow \\
\\
\text{in } A^q : \quad q_0 \dots q_k \xrightarrow{\psi'} q_{k+2} \xrightarrow{a} q_{k+3} \dots q \downarrow \\
\\
\text{in } \mathcal{M} : \quad s_0 \dots s_k \xrightarrow{\langle \boxtimes A_{q_k} \rangle \psi} s_k \xrightarrow{\psi'} s_k \xrightarrow{a} s_{k+3} \dots s \models \langle \boxtimes A_q \rangle \phi \\
\\
\text{let } v \text{ be:} \quad \dots \xrightarrow{\psi} \xrightarrow{a} \dots \\
\\
\text{in } B : \quad q'_0 \dots q'_k \xrightarrow{\psi} q'_{k+1} \xrightarrow{a} q'_{k+3} \dots q'_n \downarrow \\
\\
\text{in } \mathcal{M} \boxtimes A : \quad (s_0, q_0) \dots (s_k, q_k) \xrightarrow{\psi} (s_k, q_k) \xrightarrow{a} (s_{k+3}, q_{k+3}) \dots (s, q) \models \phi
\end{array}$$

✱

4.5 Conclusion and Future Work

Protocols are important components of *social software* [Par02] that govern the human behaviour in social interactions. In this chapter we studied the dynamics of protocols. We proposed three PDL-style logics for reasoning about protocol changes: PDL[!] handles protocol changes in the context without knowledge; PDL^{!b} extends PDL[!] with knowledge operators and Boolean tests so it can deal with the situations where events carry information according the protocols; PDL[⊠] extends the DEL framework with more general product update operations taking guarded automata as update models, which allows us to model branching protocols involving complicated tests. We showed that these three logics can be translated to PDL. What we gain is the explicitness of the language and convenience in modelling scenarios with protocol changes as we demonstrated by various examples. For interested readers who want to see more applications of the protocol changing operations, we refer to [WSvE10]

where we integrated the protocol changing operator as in PDL¹ in a specific setting of communications over channels. It is shown in [Lut06] that the public announcement logic, though equally expressive as epistemic logic, is exponentially more succinct than the pure epistemic logic in expressing certain properties on unrestricted models¹⁰. Here we conjecture that similar results apply to our logics as well. However, we leave the succinctness and complexity analysis for future work.

One thing we did not cover in this chapter is the higher-order change of protocols. For example, *I am asking you to ask her to do something* can be viewed as an announcement of a protocol concerning another protocol announcement. In the logics we presented in this chapter we did not consider protocol updates as basic events, thus excluding protocol announcements such as $!(\pi \cdot \pi')$. The exact semantics for such announcements can be complicated, and is left for future work¹¹.

Last but not least, we may introduce more operations on automata other than \boxtimes , e.g., continuation or replacement similarly to the generalized protocol announcements mentioned in the end of Section 4.2. It is interesting to see whether PDL is still closed under such extra operations and how the new operator can help to define the existing dynamic operators e.g., the various belief revision operators and preference upgrades as in [BS08a, vBL07, Liu08].

¹⁰not on S5 models as desired though

¹¹The propositional coding technique which deals with higher-order event models in [Auc09] may be useful.