Epistemic modelling and protocol dynamics

Wang, Y.

Citation for published version (APA):
Chapter 2

Preliminaries

This chapter introduces a few very basic concepts and notations which are frequently used throughout the thesis. In the follow-up chapters, we will refer to the definitions in this chapter when needed.

2.1 Finite Automata and Regular Expressions

2.1.1. Definition. (Finite Automata on Finite Words) A (non-deterministic) finite automaton is a tuple \( A = (Q, \Sigma, q_0, \rightarrow, F) \) where:

- \( Q \) is a finite non-empty set of states, with \( q_0 \in Q \) being the start state;
- \( \Sigma \) is an alphabet;
- \( \rightarrow \subseteq Q \times \Sigma \times Q \) is the set of labelled transitions over \( Q \);
- \( F \subseteq Q \) is the set of accept states.

Notation For any \( a \in \Sigma \), we write \( \rightarrow_a \) for \( \{ (q, q') \mid (q, a, q') \in \rightarrow \} \). Let \( \Sigma^* \) be the set of finite (possibly empty) strings of labels in \( \Sigma \), for any \( w = (a_0, a_1, \ldots, a_n) \in \Sigma^* \), we write \( q \vdash a \vdash \cdots \vdash a_n \vdash q' \) if there is a path \( q \rightarrow a_0 \rightarrow a_1 \rightarrow \cdots \rightarrow a_n \rightarrow q' \) in \( A \). Given an unspecified finite automaton \( A \) we use \( Q_A, \Sigma_A, q_A, \rightarrow_A \) and \( F_A \) for the corresponding components in the definition of the automaton.

Given \( \rightarrow \), we let the induced transition function \( \delta : Q \times \Sigma \mapsto 2^Q \) be defined as \( \delta(q, a) = \{ q' \mid q \vdash a \vdash q' \} \). Note that \( \delta(q, a) \) may be \( \emptyset \) for some \( q \) and \( a \). A finite automaton on finite words \( A \) is said to be deterministic, if for any \( q \in Q_A \) and \( a \in \Sigma \): \( \delta(q, a) \) is a singleton. We can extend the transition function \( \delta \) to \( \delta^* : Q \times \Sigma^* \mapsto 2^Q \) such that \( \delta^*(q, w) = \{ q' \mid q \vdash w \vdash q' \} \). It is clear that deterministic finite automata (DFA) have the property that for any word \( w \in \Sigma^* \), \( \delta^*(q, w) \) is a singleton.

Given a finite automaton \( A = (Q, \Sigma, q_0, \rightarrow, F) \) and a word \( w = (a_1, \ldots, a_n) \in \Sigma^* \), we call a sequence \( r = (q_0, q_1, \cdots, q_n) \) a run of \( A \) over \( w \) if for \( 0 \leq i \leq n \): \( q_i \rightarrow a_i q_{i+1} \).
A run \( r = (q_0, \cdots, q_n) \) is said to be accepting if \( q_n \in F \). We say \( A \) accepts \( w \) if there exists an accepting run of \( A \) over \( w \). The language of a finite automaton \( A \) is the set \( \mathcal{L}(A) = \{ w \in \Sigma^* | A \text{ accepts } w \} \). We say \( A \) and \( A' \) are language equivalent \( (A = L(A')) \) if \( \mathcal{L}(A) = \mathcal{L}(A') \).

Given an alphabet \( \Sigma \), regular expressions over \( \Sigma \) are of the form:

\[
\pi ::= 0 | 1 | a | \pi + \pi' | \pi \cdot \pi' | \pi^* \]

where \( a \in \Sigma \) and \( 0, 1 \) are constants for the empty language and the empty string respectively. We let \( \text{Reg}_\Sigma \) be the set of all the regular expressions over \( \Sigma \).

Given \( L, L' \subseteq \Sigma^* \), we define \( L \circ L' \) to be the set \( \{ wv | w \in L, v \in L' \} \). For \( n \geq 0 \) we define \( L^0 = \{ \epsilon \} \) and \( L^{n+1} = L \circ L^n \) where \( \epsilon \) is the empty string. We write \( L^* \) for \( \bigcup_{n \geq 0} L^n \).

2.1.2. Definition. (Language of Regular Expressions) The language of a regular expression \( \pi \) (denoted as \( \mathcal{L}(\pi) \)) is a set of finite strings over \( \Sigma \) defined as follows:

\[
\begin{align*}
\mathcal{L}(0) &= \emptyset \\
\mathcal{L}(1) &= \{ \epsilon \} \\
\mathcal{L}(a) &= \{a\} \\
\mathcal{L}(\pi + \pi') &= \mathcal{L}(\pi) \cup \mathcal{L}(\pi') \\
\mathcal{L}(\pi \cdot \pi') &= \mathcal{L}(\pi) \circ \mathcal{L}(\pi') \\
\mathcal{L}(\pi^*) &= (\mathcal{L}(\pi))^* 
\end{align*}
\]

The following result is well-known:

2.1.3. Theorem (Kleene’s Theorem). For any regular expression \( \pi \), there exists a finite (deterministic) automaton \( A \) such that \( \mathcal{L}(A) = \mathcal{L}(\pi) \). For any finite (deterministic) automaton \( A \) there is a regular expression \( \pi \) such that \( \mathcal{L}(\pi) = \mathcal{L}(A) \).

2.2 Kripke Models and Bisimulation

2.2.1. Definition. (Kripke Model) A Kripke model \((KM)\) is a tuple:

\[
\mathcal{M} = (S, P, \Sigma, \rightarrow, V)
\]

where:

- \( S \) is a non-empty set of states (or possible worlds);
- \( P \) is a set of proposition letters;
- \( \Sigma \) is a non-empty set of labels;
- \( \rightarrow \subseteq S \times \Sigma \times S \) is the set of labelled relations over \( S \);
- \( V : S \to 2^P \) is the valuation function.
2.2. Kripke Models and Bisimulation

We call \( P \) the vocabulary of \( M \) and \( \Sigma \) the set of labels of \( M \). \( (P, \Sigma) \) is called the signature of \( M \). A pointed Kripke model \( (M, s) \) is a KM with a designated point in the set of states. Following the tradition in modal logic, we shall call \( F = (S, P, \Sigma, \rightarrow) \) a Kripke frame.

As \( \xrightarrow{w} \) in the case of finite automata, we adapt the notion \( s \xrightarrow{a} t \) for \( a \in \Sigma \) and \( w \in \Sigma' \) in the context of Kripke models, similarly for \( S_M, P_M, \Sigma_M, \rightarrow_M \) and \( V_M \).

A Kripke model \( M \) is said to be finite, if \( S_M, \Sigma_M \) and \( P_M \) are all finite. A Kripke model is image-finite or finitely branching if for every state and every label \( a \in \Sigma \), there are only at most finitely many \( a \)-successors; it is \( \omega \)-branching if for every state and every label \( a \in \Sigma \), there are at most countably many \( a \)-successors.

An \( S5 \) Kripke model \( M \) is a KM whose labelled relations are equivalence relations, i.e., for all \( a \in \Sigma_M \) \( \xrightarrow{a} \) is reflexive \((\forall s: s \xrightarrow{a} s)\), symmetric \((\forall s, t: s \xrightarrow{a} t \iff t \xrightarrow{a} s)\), and transitive \((\forall s, t, r: s \xrightarrow{a} t \land t \xrightarrow{a} r \Rightarrow s \xrightarrow{a} r)\). Therefore, in the case of \( S5 \) models, we also use \( \sim \) to denote the set of relations. \( S5 \) models are standard models for epistemic logic where the set of labels are interpreted as the set of agents. In such a context we may use \( I \) instead of \( \Sigma \) when defining an \( S5 \) model and use \( \sim_i \) instead of \( \sim \) for \( i \in I \), following the standard notations in epistemic logic.

Note that in computer science a Kripke frame is usually called a Labelled Transition System (LTS) and Kripke models are sometimes called Kripke Labelled Transition Systems (KLTS).

2.2.2. Definition. (Bisimulation) A binary relation \( R \) between the domains of two KMs \( M = (S, P, \Sigma, \rightarrow, V) \) and \( N = (T, P, \Sigma, \rightarrow', V') \) is called a bisimulation iff \((s, t) \in R \) implies that the following conditions hold:

- **Invariance** For any \( p \in P : p \in V(s) \iff p \in V'(t) \).
- **Zig** if \( s \xrightarrow{a} s' \) in \( M \) then there exists a \( t' \) in \( N \) such that \( t \xrightarrow{a'} t' \) and \( s'Rt' \).
- **Zag** if \( t \xrightarrow{a'} t' \) in \( N \) then there exists an \( s' \) in \( M \) such that \( s \xrightarrow{a'} s' \) and \( s'Rt' \).

Two pointed Kripke models \( (M, s) \) and \( (N, t) \) are said to be bisimilar \( (M, s \sim N, t) \) if there is a bisimulation \( R \) between them such that \((s, t) \in R \). We say a bisimulation \( R \) is total, if every world in one model is linked by \( R \) to some world in the other model. We write \( M \sim N \) if there is a total bisimulation between \( M \) and \( N \).

Note that the above standard bisimulation is defined between models with the same signature. In this thesis we will also work with models with different vocabularies. We say two pointed models \( (M, s) \) and \( (N, t) \) are restricted bisimilar w.r.t \( P' \subseteq P_M \cap P_N \) (notation: \( M, s \sim_{P'} N, t \)), if \( M, s \) and \( N, t \) are bisimilar with the original invariance condition replaced by a restricted invariance condition:

- **[P’-Invariance]** for any \( p \in P' : p \in V_M(s) \iff p \in V_N(t) \).
Similarly we can define total restricted bisimulation w.r.t $\mathcal{P}' (\mathcal{M} \subseteq_{\mathcal{P}} \mathcal{N})$ in the straightforward way.

Note that an autobisimulation of a model is an equivalence relation on the state space of a model. Thus we can have a quotient model w.r.t to the maximal autobisimulation on a model.

2.2.3. Definition. (Bisimulation Contraction) 
Given a Kripke model $\mathcal{M} = (S, P, \Sigma, \rightarrow, V)$, let $\equiv_b \subseteq S \times S$ be the autobisimulation: $\{(s, t) \mid \mathcal{M}, s \equiv \mathcal{M}, t\}$. The bisimulation contraction of $\mathcal{M}$ is the quotient model $\mathcal{M}/\equiv_b = (S', P, \Sigma, \rightarrow', V')$ where:

- $S' = \{[s] \mid s \in S\}$ where $[s]$ is the equivalence class containing $s$ w.r.t $\equiv_b$;
- $\{[s], a, [t]\} \in \rightarrow'$ iff $(s, a, t) \in \rightarrow$;
- $V'(\{s\}) = V(\{s\})$.

We can adapt the definition of bisimulation for finite automata by replacing the invariance condition with the following:

[Accept Invariance] : $s \in F \iff t \in F'$

where $F$ and $F'$ are the sets of accept states in two automata. We say automata $A$ and $B$ are bisimilar if there is a bisimulation $R$ between $Q_A$ and $Q_B$ with the accept invariance condition such that $(q_A, q_B) \in R$. It is easy to see that $A \equiv A' \implies A \equiv_L A'$, but the converse does not hold.

2.2.4. Definition. ($n$-round Bisimulation Game) 
An $n$-round bisimulation game $\mathcal{G}_n(\mathcal{M}, s, (N, t))$ between two pointed KMs $(\mathcal{M}, s)$ and $(N, t)$ with the same signature is a two player game based on the configurations in $S_M \times S_N$. The initial configuration is $(s, t)$ and the players, Spoiler and Verifier, play in rounds. Each round consists of two moves: first by Spoiler and then by Verifier. At each configuration $(s', t')$, there are two options:

- Spoiler selects an $a \in \Sigma$ and a state $s''$ in $\mathcal{M}$ such that $s' \xrightarrow{a} s''$ and then Verifier needs to come up with a state in $\mathcal{N}$ such that $t' \xrightarrow{a} t''$ and $V(s'') = V(t'')$. The configuration is then changed to $(s'', t'')$.
- Spoiler selects an $a \in \Sigma$ and a state $t''$ in $\mathcal{N}$ such that $t' \xrightarrow{a} t''$ and then Verifier needs to respond with a state in $\mathcal{M}$ such that $s' \xrightarrow{a} s''$ and $V(s'') = V(t'')$. The configuration is then changed to $(s'', t'')$. 
Spoiler wins the game if within \( n - 1 \) rounds some configuration \((s', t')\) is reached such that Spoiler can make a legal move but Verifier does not have a legal move to respond. Verifier wins the game otherwise.

We say \((M, s)\) and \((N, t)\) are modally equivalent \((M, s \equiv_{ML} N, t)\) if \(M, s\) and \(N, t\) satisfy exactly the same basic modal logic (ML) formulas\(^1\). The following facts are well known (cf., e.g.,\([BdRV02]\)).

2.2.5. Fact. For image-finite pointed Kripke models \((M, s)\) and \((N, t)\), the following are equivalent:

- \(M, s \leftrightarrow N, t\).
- \(M, s \equiv_{ML} N, t\).
- for all \(n \in \mathbb{N}\) : Verifier has a winning strategy in the game \(G_n((M, s), (N, t))\).

2.3 Three Logics

2.3.1 Propositional Dynamic Logic

Propositional Dynamic Logic (PDL), introduced by Fischer and Ladner [FL79] (following the idea of [Pra76]), is a branching-time logic of programs (represented by regular expressions):

\[
\phi ::= \top | p | \phi \land \psi | \neg \phi | \langle \pi \rangle \phi
\]

where \(p\) ranges over a set of propositions \(P\) and \(\pi\) is a regular expression over some alphabet \(\Sigma\) with tests in terms of PDL formulas:

\[
\pi ::= 0 | 1 | a | ?\phi | \pi + \pi | \pi \cdot \pi | \pi^* \]

where \(a \in \Sigma\). When \(\Sigma\) is not fixed, we use \(\text{PDL}_{\Sigma}\) to denote the PDL language based on \(\Sigma\). As usual, we define \(\perp, \phi \lor \psi, \phi \rightarrow \psi\) and \([\pi]\phi\) as the abbreviations of \(\neg \top, \neg(\neg \phi \land \neg \psi), \neg \phi \land \psi\) and \(\neg(\neg \pi)\neg \phi\) respectively.

Intuitively, \([\pi]\phi\) says that there is an execution of program \(\pi\) such that after the execution \(\phi\) holds.

We define the satisfaction relation \(\models\) between a pointed model \((M, s)\) with the signature \((\mathcal{P}, \Sigma)\) and a PDL\(_{\Sigma}\) formula \(\phi\) as follows:

\[
\begin{align*}
M, s \models p & \iff p \in V_M(s) \\
M, s \models \neg \phi & \iff M, s \not\models \phi \\
M, s \models \phi \land \psi & \iff M, s \models \phi \text{ and } M, s \models \psi \\
M, s \models [\pi]\phi & \iff \exists s' : [\pi][s'] \text{ and } M, s' \not\models \phi
\end{align*}
\]

where \([\pi]\) is defined as:

\(\text{ML}\) extends propositional logic with modal formulas \(\Box \phi\) and their Boolean combinations.
Chapter 2. Preliminaries

\[\begin{align*}
&\mathcal{M}, s \models \langle \pi \rangle \phi \iff \text{there exists a path } s = s_0 \langle a_1 \rangle s_1 \langle a_2 \rangle \cdots \langle a_n \rangle s_n \text{ in } \mathcal{M} \text{ such that } \\
&\quad \mathcal{M}, s_n \models \phi \text{ and } a_0 a_1 \ldots a_n \in L(\pi)
\end{align*}\]

PDL can be axiomatized by the following axioms and inference rules \[\text{Seg82, Par78}\]

<table>
<thead>
<tr>
<th>Tautology</th>
<th>All the tautologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>([\pi](\phi \rightarrow \phi') \rightarrow ([\pi]\phi \rightarrow [\pi]\phi'))</td>
</tr>
<tr>
<td>0</td>
<td>([0]\phi \leftrightarrow \top)</td>
</tr>
<tr>
<td>1</td>
<td>([1]\phi \leftrightarrow \phi)</td>
</tr>
<tr>
<td>Test</td>
<td>(?\psi\phi \leftrightarrow (\psi \rightarrow \phi))</td>
</tr>
<tr>
<td>Seq</td>
<td>(\langle \pi_1 \cdot \pi_2 \rangle \phi \leftrightarrow (\langle \pi_1 \rangle \langle \pi_2 \rangle \phi))</td>
</tr>
<tr>
<td>Or</td>
<td>(\langle \pi_1 + \pi_2 \rangle \phi \leftrightarrow (\langle \pi_1 \rangle \phi \lor \langle \pi_2 \rangle \phi))</td>
</tr>
<tr>
<td>Star1</td>
<td>(\langle \pi^* \rangle \phi \leftrightarrow (\phi \lor \langle \pi \rangle \langle \pi^* \rangle \phi))</td>
</tr>
<tr>
<td>Star2</td>
<td>([\pi^<em>](\phi \rightarrow [\pi]\phi) \rightarrow (\phi \rightarrow [\pi^</em>]\phi))</td>
</tr>
</tbody>
</table>

Rules
- \(\because \phi \rightarrow [\pi]\phi\)
- MP: \(\frac{\phi, \phi \rightarrow \psi}{\psi}\)

Note that with the presence of tests \(?\phi\) we can eliminate basic programs 0 and 1 by defining them as \(?\bot\) and \(?\top\) respectively. Sometimes we are interested in the test-free fragment of PDL in which we do not have \(?\) as one of the program constructors but we do have 0 and 1.

We write \(K_i \phi\) (i knows \(\phi\)) and \(\overline{K}_i \phi\) (i thinks \(\phi\) is possible) for \([i]\phi\) and \(\langle i \rangle \phi\) respectively, when interpreting PDL on \(S5\) models in an epistemic setting. We write \(C_G \phi\) (\(\phi\) is common knowledge among the agents in \(G\)) as \(\{(i_1 + \cdots + i_n)\} \phi\) if \(G = i_1, \ldots, i_n \subseteq I\).

\[\text{Seg82, Par78}\]

\[\text{Note that with the presence of tests } ?\phi \text{ we can eliminate basic programs 0 and 1 by defining them as } ?\bot \text{ and } ?\top \text{ respectively. Sometimes we are interested in the test-free fragment of } \text{PDL in which we do not have } ? \text{ as one of the program constructors but we do have 0 and 1.}\]

\[\text{The PDL formulas which are valid (i.e. hold on all the pointed models) are precisely the ones that can be derived from the following proof system.}\]
2.3.3 Dynamic Epistemic Logic

A different perspective on the dynamics of multi-agent system is provided by the development of so-called Dynamic Epistemic Logic (DEL) [Pla89, GG97, BMS98]. The focus of DEL is not on the temporal structure of the system but rather on the epistemic impact of the events as the agents perceive them. The following PDL-style DEL language is based on the exposition in [vBvEK06]:

\[ \phi := \top | p | \neg \phi | \phi_1 \land \phi_2 | \langle \mathcal{A}, e \rangle \phi | \langle \tau \rangle \phi \]

where \( \mathcal{A} \) is an event model defined below with \( e \) as its designated event.
2.3.2. Definition. (Event Model) An event model \( A \) is a tuple:

\[
A = (E, \Sigma, \rightarrow, \text{Pre})
\]

where:

- \( E \) is a finite non-empty set of events.
- \( \Sigma \) is a set of labels.
- \( \rightarrow \subseteq E \times \Sigma \times E \) is the set of labelled transitions.
- \( \text{Pre} : E \mapsto \text{Form}(\text{DEL}) \) where \( \text{Form}(\text{DEL}) \) is the set of \( \text{DEL} \) formulas. Intuitively, \( \text{Pre} \) assigns to each action a precondition in the form of a \( \text{DEL} \) formula that can be constructed in an earlier stage of the inductive definition of the language.

Notation In the epistemic setting, the relations \( \rightarrow_i \) in the action model are assumed to be equivalence relations, thus we may use \( \leftrightarrow_i \) to denote them. \( \leftrightarrow_i \) models agent \( i \)'s observational power on events in \( E \) (e.g. \( e_1 \leftrightarrow_i e_2 \) means agent \( i \) can not distinguish event \( e_1 \) and \( e_2 \)).

The semantics for PDL formulas is as usual and for \( \langle A, e \rangle \phi \):

\[
\models_{M, s} \langle A, e \rangle \phi \iff M, s \models \text{Pre}(e) \text{ and } M \otimes A(s, e) \models \phi
\]

where \( \otimes \) is defined as follows:

2.3.3. Definition. (Product Update \( \otimes \)) Given a Kripke model \( M = (S, \Sigma, \rightarrow, V) \) and an event model \( A = (E, \Sigma, \rightarrow, \text{Pre}) \), the product model \( (M \otimes A) = (S', \Sigma, \rightarrow', V') \) where:

\[
\begin{align*}
S' &= \{(s, e) \mid M, s \models \text{Pre}(e)\} \\
\rightarrow' &= \{(s', e', e) \mid s \rightarrow s' \text{ and } e \rightarrow e'\} \\
V'(s, e) &= V(s)
\end{align*}
\]

The simplest event model is perhaps the one modelling a public announcement of \( \phi \) (notation: \( !\phi \)) depicted as the following event model \( A_{!\phi} \):

\[
e : (I, I)\]

where \( \phi \) is the precondition of this singleton model, and \( \rightarrow_i \) denotes the reflexive relations for each \( i \in I \). Let \( M_{!\phi} \) be the Kripke model \( (S, P, I, \sim, V) \) where:

- \( S = \{s \in S_M \mid M, s \models \phi\} \)
- \( \sim = \sim_M \cap (S \times \Sigma \times S) \)
2.3. Three Logics

- \( V = V_M \) (i.e. the restriction of \( V_M \) on the domain \( S \)).

It is easy to see that updating \( A_\phi \) on a static model \( M \) amounts to restricting the \( M \) by the states which satisfy \( \phi \): \( M \otimes A_\phi \models M_\phi \).

As a simple yet important fragment of DEL, the Public Announcement Logic (PAL) \([\text{Pla}89, \text{GG}97]\) is usually presented as follows:

\[
\phi ::= \top \mid p \mid \neg \phi \mid \phi_1 \land \phi_2 \mid K_i \phi \mid [!\psi] \phi
\]

where \( K_i \phi \) and \([!\psi] \phi \) are equivalent to \([i] \phi \) and \([A_\psi] \phi \) in DEL respectively.

As for the expressivity of DEL, \([\text{vBvEK}06]\) showed that adding product updates to PDL does not increase the expressive power of PDL:

**2.3.4. Fact.** \([\text{vBvEK}06]\) For any DEL formula \( \phi \) there is a PDL formula \( \phi' \) such that for all pointed Kripke models \( M, s : M, s \models_{\text{DEL}} \phi \iff M, s \models_{\text{PDL}} \phi' \).