Modeling Nomic in LKIF-Core ontology.

Marc Bron, Abdallah El-Ali, Xingrui Ji, Szymon Klarman

September 29, 2007

Leibniz Center for Law

Contents

This report, part of ESTRELLA project, presents results of the work on building an OWL model of Nomic rules, based on the LKIF-Core ontology. The primary aim of the task was to test capabilities of the LKIF-Core in modeling a sample piece of legislation. In the report we discuss the structure of the proposed representation, explain the modeling approach and elucidate the role of LKIF-Core in the model.

1 Introduction

1.1 LKIF-Core

LKIF-Core is an upper- or core ontology of legal terms, essentially intended to support top-down knowledge acquisition and representation of domain ontologies. Despite that an upper ontology such as LKIF-Core serves the purpose of providing the initial structure (comprised of super-classes) for a given domain ontology (such as the game of Nomic, as we shall see below), it is often the case that the concepts defined are highly abstract and relatively legal-domain independent. The highly abstract concepts, however, allow one to use LKIF-Core (as it covers the basic concepts of law) to assess similarities and differences in particular legal domains of differing jurisdictions, while simultaneously not being exclusively for or about law.

In what way can LKIF-Core be used, or rather, what role does such an upper ontology play? In the context of modeling the game of Nomic, we place emphasis on two primary roles of LKIF-Core (see Deliverable 1.4, [1], for a detailed account of the different uses of upper/core ontologies): To organize and structure information, and to do reasoning and problem-solving.

The first role, as briefly mentioned earlier, concerns pre-structuring the development of a domain ontology, by allowing for top-down knowledge acquisition to subsume most or all domain-specific terms as sub-classes of the more abstract concepts defined initially in the upper ontology. For example, the definition of norm in LKIF-Core serves as a broad concept that subsumes specific norms in some legal domain.

The second role concerns the representation of knowledge in a domain by which automated reasoning is made possible, by representing problems and generating solutions for those problems. The ontology in this case functions as a terminological part of a knowledge base that enables understanding of assertions made about the problem situation to be solved. For example, one is able to write the relationships between defined terms that provide deontic qualifications such as obligations and prohibitions in norms, allowing the reasoner to use these axioms to identify violations in a regulation that contains these norms. How LKIF-Core assumes these roles in practice and what the top-level concepts are will be further discussed in Section 3.

1.2 The Game of Nomic

Nomic (from the Greek $\nu\acute{o}\mu o\zeta$ (nómos), which means law) is an abstract game of rule-making and legislation, that was first invented by Peter Suber in 1980 (Vreeswijk, [2]). The central idea behind Nomic is to change the rules of Nomic. The self-modifying game of Nomic could be thought of as a complete microcosm of a functioning legal system. The parallel between Nomic and a legal system holds at the level of rules in the game, where changing the rules of the game itself is a move in Nomic. Nomic, a two-tiered system, designed as such for the sake of simplicity, contains an initial set of rules that contains on the one hand immutable rules, which govern more basic processes and are thus more difficult to change, and mutable rules, which are more susceptible to change. For example, while a mutable rule can immediately be amended during a given turn by a player, this is not so for an immutable rule, which first requires a player to transmute it into a mutable rule before further change can take place.

In modeling the game of Nomic in LKIF-Core, two distinctions in the modeling approach are to take effect: modeling Nomic as a multi-player game qua game, and modeling Nomic as a piece of legislation. The former taps on the notion that a game has a sequential ordering of events which must be modeled, while the latter surfaces the difficulty in modeling the game rules as norms, that allow or disallow certain actions or situations to occur. To elucidate, consider how in each turn the rules of the game change. These changes must be represented temporally taking into account the history of each player's moves and representing it at a current moment in time.

For example, Rule 202 explains the basic setup for playing: For a given turn, a player can propose a rule change and have it voted on, and if the vote succeeds, the change will be implemented the following turn. Since this rule can be changed, the game is (potentially) no longer the same game the players started with, since the game has now differing deontic criteria with respect to voting, rule changes, or whatever the new self-modifying rule that was voted on now dictates. This essentially highlights the legislative features inherent in the self-reflexive game of Nomic, and modeling it thus requires maintaining a balanced understanding of the game as an abstract game played by a number of players, and a continuously changeable legal game where the laws in the game are ultimately subject to change if so wished by the players.

2 Overview of Taxonomy

When building an ontology of a certain domain for a particular purpose, the first step is to identify the main concepts. More concepts can later be added in order to add detail to the model. The amount of detail depends on the domain and the purpose of the ontology. Modeling the entire game of Nomic, although an interesting goal in itself, is too huge a task for the purposes of this project. Instead, a number of core, most representative elements from the game of Nomic have been selected.

To be able to use Nomic as an example of how legislation could be modeled, the game has to be stripped to its core. It is important that this core keeps its characteristics that make Nomic a good example for modeling legislation. The concepts that have been selected capture these characteristics.

Rules: Every game consists of rules — without rules, there is no game. From this initial set of rules the core concepts in the game can be identified. There are for example players, turns, actions and things manipulated by actions. Besides concepts, rules also define the status of rules e.g. that rules can be mutable or immutable.

In the model, a rule is represented as an individual of the class `Rule`. This individual is linked to a specific part of the model that represents what is expressed by the rule. The link is the qualification of this part by the rule, either allowing or disallowing it explicitly or allowing it simply by stating its existence e.g. the concept of a player is never explicitly allowed, but used everywhere. The separation between a rule and what is regulated by that rule allows statements to be made about rules in general or about individual rules without changing the implementation of the rule. Some concepts in the model are regulated by multiple rules. In this case, all these rules contribute a little to that concept and will be linked to it.

Players: To play the game it is necessary to have players - without them, Nomic might still be considered a game, albeit a game that is never played. The concept of a player is introduced as a role that is played by a person who plays the game. Differentiating between a person and its role is needed because during the game the concept of player can change, while the person playing the game will remain the same.

Players are modeled as individuals under the class `Player` and linked to individuals under the class `Person`. To allow playing the game, a person has to be a player. What kind of conditions have to be met to be a player is hard to define e.g. are you still a player if you cheat, do not intend to play the game or plan to sabotage the game? Usually there will be some agreement between the people planning on playing a game, on who the players will be and thus, this information can just be asserted into the model.

Turns: The period during which a game is played, is usually divided in a number of turns. A turn is then a limited period of time used by a player or multiple players to perform some actions. As actions change the state of the game, the step from a turn to the next one represents the change of one state to another. So turns actually refer to certain states and impose an order on them. This can be used to keep track of time in the game. For example, if a state that exists in a certain turn changes, then the next turn refers to the resulting state, while the previous turn still refers to the old state. A turn can also be divided into smaller parts e.g. a proposing phase, a voting phase, and a scoring phase. To reduce the complexity of the game however, only whole turns are considered.

When modeling Nomic, there is a turn which is the current turn that defines the current model of the game. The history of the game is known until the current turn, which is then also the last (known) turn. The previous states can be revisited by changing the current turn to an older turn, hence referring to an older model. The last turn however will still be the last known turn. The link between the turns and parts of the model will be further explained in section 3.4.

Actions: The things that give a game its character are the actions that are allowed by rules. The action that best characterizes Nomic is the ability to change the rules. Most of the other actions are needed to create the requirements to perform a rule change. Therefore we decided to model only two actions: 1) changing a rule, i.e. amending, enacting, transmuting or repealing a rule, and 2) voting on a proposed rule change.

The other actions are modeled implicitly in that their results have to be asserted into the model. Those results will serve as requirements for the modeled actions. For example, while the action of proposing a game proposal is not included in the model, an appropriate instance of a proposal, however, has to exist before a rule change can be performed.

3 Model and Modeling Approach

The basic goal of the project is to extract and formalize the ontology of Nomic's concepts, as found in the Initial Set of Rules, and embed it in the LKIF-core ontology of basic legal terms. While approaching the task we have encountered some difficulties, stemming from a very uncommon character of Nomic, which, though not directly related to the original goal, seemed to be very interesting and potentially of some significance to Knowledge Representation practice in general. We therefore decided to address them as well, and incorporate the proposed solutions into our model. The two supplementary goals that directed our modeling approach along with the primary one are:

1. To develop a consistent and reasonably efficient strategy for handling changes in the represented knowledge, which may take place during a game, while retaining the ability of referring to the relevant parts for the different stages of the game.

2. To establish a practical approach to modeling actual game scenarios and provide some convenient means of validating consecutive stages of a game both while modeling a game in progress and after completing a whole scenario.

In the following subsections we will first give a detailed account of the structure of the ontology, justifying each of our choices, indicating the links to concepts in LKIF, and explaining the general modeling approach with a special emphasis on the role of the reasoner. Then we will turn to the issues pertaining to the problem of handling the knowledge changing over time.

3.1 Methodology

The approach taken in building the model can be described as a compile-and-debug method, as often seen during programming. An even better description in this case would be a create-and-classify method. Only checking the definition of newly added concepts by asserting the type and checking if the ontology is consistent, is a less stricter way of checking the definition of a concept than to let the reasoner infer the type of concept by classification.

This approach works as follows: every time a concept is defined with necessary and sufficient conditions, the reasoner is ran to check if the concept gets classified correctly i.e., placed in the correct place in the ontology. If this is so the specific subclass relation is asserted. Next, an individual for this concept is created and the reasoner is again ran to check if the individual gets classified under the correct concept (see also section on Modeling Scenarios).

In the approach adopted, there is a strong reliance on the role of the reasoner while building the model. To keep this process tractable, the reasoning process should not take too long. Since the LKIF ontology in which the Nomic concepts are placed is by itself a large and complex ontology, it takes some time to complete the reasoning. When the Nomic concepts are added to this, the reasoning time quickly becomes intractable for the purpose of checking concept definitions. To reduce the reasoning time, a stripped down version of LKIF was created: LKIF Skeleton.

3.2 LKIF Skeleton

The quick growth in reasoning time in LKIF is partly caused by the fact that none of the concepts are disjoint. Disjointness reduces reasoning time because an individual that gets classified under concept A does not have to be matched to any other concept that is disjoint with A. If disjointness between two concepts is asserted very high up in the ontology, large branches of the concepts do not have to be searched. As the top of the ontology is defined by LKIF and it has no disjointness between its concepts, adding Nomic concepts to it results in very large branches of individuals and concepts.

In LKIF Skeleton only the concepts from LKIF-Core that are relevant for the modeling of Nomic are used, because all the relations and concepts in LKIF that are needed in the top of the ontology can be made disjoint. The LKIF Skeleton top consists of the following concepts:

Agent: is disjoint with all the other concepts in LKIF Skeleton. In LKIF this is not the case because an agent can also serve as a medium for beliefs or intentions. An agent having some belief would then be classified as a medium e.g. the bearer of some belief. As beliefs are not used in the Nomic core, Agent and Medium are made disjoint. An agent can also be an actor in an action or play a certain role.

Qualified: is disjoint with Agent and Abstract_Concept. Something that is qualified (allowed or disallowed) by some norm, will be classified as Qualified. Actions can be allowed or disallowed by norms, and so in order to get classified as Qualified, this concept cannot be disjoint with Change, as actions are a subconcept of Change.

Change: is disjoint with all concepts except Qualified. The concept of Change holds dynamic concepts i.e., processes as a subclass of Change, and actions as a subclass of Process. The Action concept requires at least one actor to participate in the action. Besides actors, an action can have relations to other concepts of which the most important are: Requirements, Resources and Results. These relations to other concepts can define a certain action.

Abstract_Concept: only holds the concept of Time Interval and is disjoint with all other classes.

Mental_Concept: is disjoint with Agent, Abstract_Concept and Change and holds many important concepts: Role, Norm, Proposition and Expression.

A Role is a direct subclass of Mental_Concept and it can be related with an Agent. Associated with a Role is certain behavior i.e. actions that are allowed for the agent playing the role.

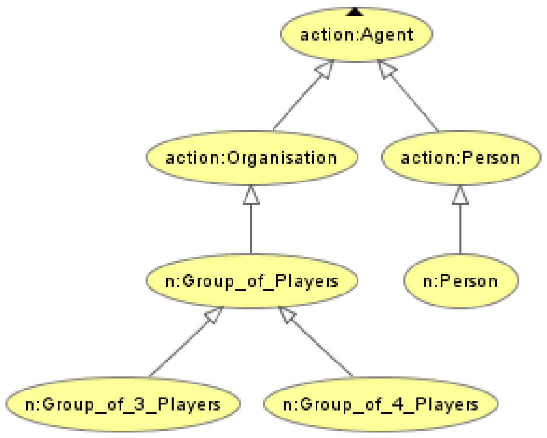
Mental_Object is another direct subclass of Mental_Concept. It holds the concept of a proposition and a qualification. A proposition is the content of an assertion. If a proposition exists on some medium i.e., a document, then it is an expression.

A qualification expresses a judgement about some thing, where this thing could be a proposition. Norm is a subclass of Qualification and the content of a norm qualifies things by allowing or disallowing it.

3.3 Nomic in LKIF Skeleton

Now that the interpretation and intended use of the concepts of LKIF Skeleton have been introduced, the placement of Nomic concepts in this ontology can be explained.

Agent and Person: An agent who plays certain roles is allowed to take certain actions. A person is an agent. A person who plays the role of a player is allowed to take actions in Nomic. Besides individual people, organizations can also be agents. In Nomic, a group of players can vote on some proposal. That group then has to consist of only those people that have the role of player.



Agents can perform actions — the most important property that relates agent concepts to other concepts in Nomic is the `actor\_in` property. It asserts that a person or group is an actor in a certain action.

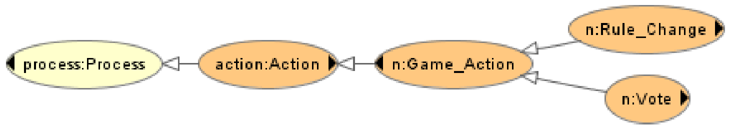
Action, Rule_Change and Vote: A process is some kind of change and actions are some kind of processes. An action requires at least an actor — someone to perform the action. Other properties are requirement, something that is needed for the action to occur, a resource, something that is used up during the action and a result, which is something brought about by the action.

As an example, consider the concept `Vote` that is a Nomic action. It has a requirement relation to a `Game\_Proposal`, which means that there has to be a proposal before a vote can take place. The result of this vote action is some `Voting\_Status`, that holds the outcome of the vote. Finally there has to be an agent, in this case a group of players that are also voters, who are able to perform the vote action.

The other main action in Nomic is a `Rule\_Change`, it consists of the following four actions: `Repeal`, `Transmutation`, `Enactment` and `Amendment`. Each of these actions are defined by their properties: requirements, results, etc...

For the `Amendment` action the resource is a `Rule` that is mentioned in an `Amendment_Proposal` as being the subject of an amendment. This rule does not exist anymore after the action. The result is a `Rule` that is proposed by an `Amendment_Proposal`. This action further requires an `Amendment_Proposal` that proposes and amends the already mentioned rules. The action is performed by some actor that is a `Person` that plays a `Game_Role`. This last property is actually inherited from `Game_Action`.

Note that `Amendment_Proposal` is mentioned three times in the properties



of an `Amendment`. It is intuitively clear that this should be the same proposal, for a reasoner however this is not the case. There could for example be two different instances of `Amendment_Proposals`, each amending and proposing different rules. In such case the first proposal could be used for the result and the second one for the requirement. The rule that is proposed by the first proposal is then different from the rule that is the result of the action.

In principle, this problem could be solved only by use of bound variables, not allowed by OWL DL syntax. To get around this, the assumption that there can only be one proposal every turn has to be made. This is, at least at the start of the game, in compliance with the rules. Every proposal can then be related to a unique `Game_Turn` and this makes it possible to identify individual proposals. This technique can of course be extended to other concepts.

Qualified Concepts: As stated before, there is some kind of judgement about a `Qualified` concept. A `Voting_Status` is the result of a `Vote` action, the `Pass_Criterion` will determine the result of the `Vote` and qualify the `Voting_Status` as being passed (`Pass`) or rejected (`Reject`). In the same way, a role of `Winner`, played by some person, gets classified under `Qualified` since it is qualified by some `Winning_Criterion`.

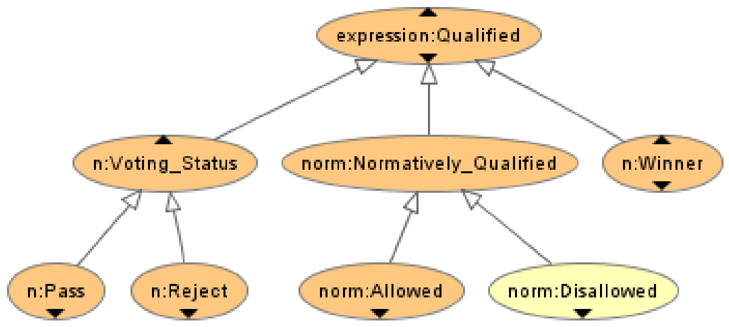
The expression of a judgement is more clearly seen in the context of the concept `Normatively_Qualified`. Some thing gets classified under this concept if it is allowed or disallowed by some norm. A norm in the context of Nomic is a `Rule` and a rule determines what is allowed and disallowed in Nomic. For the reasoner to determine what actions are allowed, it checks all the concepts (in most cases, actions) that are allowed by the rules and classifies them under the concept `Allowed`, or `Disallowed` if actions are explicitly forbidden.

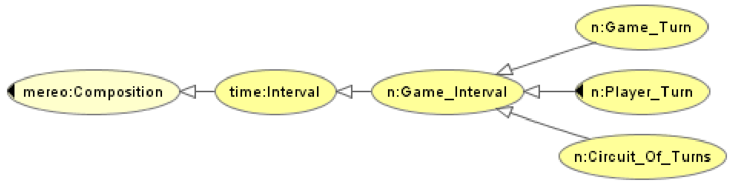
In this way a meta rule like Rule 101 can be modeled. It states:

"All players must always abide by all the rules then in effect, in the form in which they are then in effect..."

If a person that plays the role of player is restricted only to being an actor in actions that are allowed, then the person cannot do anything that is disallowed and so will follow all the rules or make the ontology inconsistent.

Time and Game_Turn: The concept of `Game_Turn` was already used to distinguish between different proposals. It is a subclass of `Game_Interval`, which



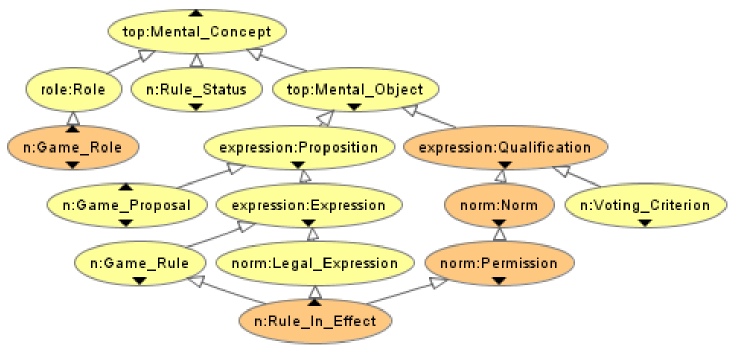


is a subclass of `Time:Interval`. `Time:Interval` is an LKIF concept that holds periods of time. A `Game_Interval` is a period of time during a game and this could be the entire game, just a single turn or a `Circuit_Of_Turns`. The latter concept is a round of turns where each player gets to act. A `Player_Turn` relates a `Game_Turn` to a specific player. The further use of turns however, will be explained in section 4.

Mental_Concept, Game_Role, Game_Rule and Qualification:
A `Game_Role` is a subclass of the `Role` concept. Different roles allow agents to be involved in different actions. A person playing the `Player` role is allowed to do Nomic actions. The `Voter` role allows a person to vote and is equivalent to `Player` as defined in the initial set of Nomic rules. The `Winner` role is played by the person that has the most points at the end of the game.

A `Game_Proposal` is required to perform rule changes - it is a subclass of `Proposition` because it does not have to exist in writing. A `Game_Rule` however has to be in writing and is therefore a subclass of `Expression`. Now a `Game_Rule` does not have to be in effect, it can be an old rule. A rule that is in effect, however is a subclass of `Legal_Expression` and falls in the concept `Rule_In_Effect`. This concept is also a subclass of `Permission`, which is a subclass of `Norm`, which is a `Qualification`. Because rules are norms they can qualify actions, and the actions allowed by the rules in effect can be classified as `Allowed` or `Disallowed`.

A problem with this approach is that it is not certain if all the rules in effect are found. In the initial set there are 29 rules, if a rule is valid in the current turn of the game and qualifies a qualification then it will be classified as a



`Rule_In_Effect`. When only two rules are classified as rules in effect, then this is considered the initial set. There is no way of checking if the correct number of rules is found. So it has to be assumed that all the rules that are classified under `Rule_In_Effect` are the initial, or current set of rules. This is only true if all the rules have been asserted to the model correctly and this task is placed entirely on the user.

3.4 Temporal Aspects of Nomic

One of the challenging problems in modeling Nomic is devising a suitable representation of its temporal aspects. Just like any game, Nomic also can be interpreted in terms of a sequence of moves made by players in consecutive turns. Apart from this basic similarity, however, Nomic is essentially different from other games. The terminological knowledge underlying a typical game is encoded in a fixed set of rules which remain constant throughout the time of playing. On the contrary, since each move in Nomic is (at least initially) intended as a change of some of its rules, and so possibly some of the terminological knowledge expressed by them, an adequate representation of Nomic should account for these changes and allow for determining which part of knowledge applies in particular turns. Consider for example the following fragment of Nomic's Initial Set of Rules:

> Rule 105
>
> Every player is an eligible voter. [...]

Rule 105 establishes the equivalence relation between concepts `Player` and `Voter`, which could be straightforwardly expressed as a TBox axiom in the Nomic's ontology:

$$\text{(R105) } \texttt{Voter} \equiv \texttt{Player}$$

However, in the course of a game, a group might want to amend Rule 105, for instance into:

Rule 105*

Every player is an eligible voter, except when it is his turn. [...]

As a result axiom (R105) is no longer valid and the relation between the two concepts should rather be expressed as:

$$\text{(R105*)}\ \texttt{Voter} \equiv \texttt{Player and TR}$$

where `TR` is the additional restriction. Clearly, denotation of `Voter` changes due to the amendment, enforcing a revision in the underlying ontology. Taking a simple *update* approach to modeling Nomic, which would require revising the ontology every time a change occurs, though intuitively correct, would nevertheless be not satisfying if one wants to maintain the ability of reasoning over whole game scenarios, i.e., of determining whether in each turn the game was played according to the rules that were then in effect. Having this goal in mind we have adopted a more complex approach to handle the temporal representation of Nomic, which we describe below.

Proposed Representation

The approach employs a three-layer formalism consisting of *time stamps* on individuals and *time scopes* of concepts, augmented by the *focusing role* of the `Current_Turn` individual.

First of all, we have introduced the class `Game_Turn`, whose instances comprise the basic reference system for all temporal reasoning. Game turns are linearly ordered by two LKIF properties `time:before` and its inverse `time:after`, forming a discrete time line.

The *time stamping* technique, which we applied in our approach, comes down to establishing links between relevant individuals (these are in fact instances of the majority of the ontology's concepts) and game turns, this way marking their relative positions on the time line and accounting for the sequential character of the game. For this purpose we use two relations with a range restricted to `Game_Turn`:

$$\texttt{applicable\_from} \subseteq \texttt{owl:Thing} \times \texttt{Game\_Turn}$$
$$\texttt{applicable\_to} \subseteq \texttt{owl:Thing} \times \texttt{Game\_Turn}$$

All concepts whose instances require a temporal reference to the time line are expected to satisfy the following restriction:

$$\text{(TSt)}\ \texttt{(applicable\_from exactly 1) and (applicable\_to exactly 1)}$$

For the clarity of representation we have used (TSt) as the only necessary and sufficient condition of the class `Time_Stamped`, which is asserted as an additional superclass to all appropriate concepts.

The intended common-sense meaning of the applicability relations varies depending on the types of arguments. The assertion:

```
applicable_from(instance, game_turn_x)
  applicable_to(instance, game_turn_y)
```

should be paraphrased as:

- `instance` *is in effect starting from* `game_turn_x` *until* `game_turn_y` — whenever the `instance` is a game rule,
- `instance` *is played starting from* `game_turn_x` *until* `game_turn_y` — when it is a role played by some actor,
- `instance` *takes place in* `game_turn_x` — if the `instance` is an action, or some kind of speech act (like expressing a judgment on the result of a voting or a statement of a rule change proposal) — in general any entity not extensive in time[1].

The relations are to be interpreted as inclusive, i.e. whenever an individual is `applicable_from` (or `applicable_to`) some game turn, then it is applicable also in that turn. Moreover we assume that all assertions will satisfy the following constraint:

if `applicable_from(i, x)` and `applicable_to(i, y)`
then `x` is `time:before y`[2]

Parallel to time stamps on individuals we have introduced *time scopes* on those concepts that are amenable to change during the game. This restriction represents the scope of game turns during which a given piece of terminological knowledge is valid. Assume for example that the amendment of Rule 105, described above, took place with the end of the second turn. We would say that the definition of `Voter` based on the axiom (R105) was valid from turn 1 to turn 2, and the other one, specified by (R105*), from turn 3 on.

The general idea of the time scope restriction can be formally expressed as follows. Let `TSc` be a time scope of some concept defined as:

`TSc` $\equiv$ `((time:after has Begin_Turn) and (time:before has End_Turn))`

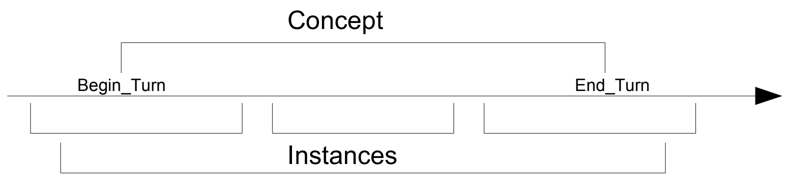
[1]The value of `applicable_to` relation should in this case be always identical to this of `applicable_from`.

[2]In order to shorten some of the largely used temporal restrictions, and thus ease their human readability, we have imposed reflexivity of `time:before` and `time:after` properties on all game turns. An expression `x` is `time:before y` is hence interpreted as `x` is (strictly) before `y` or `x=y`. Whenever it is necessary to refer to instances strictly before or after some point of time (which is very rare in the model) we can employ time`time:immediatly-before` and `time:immediatly-after` which are left as irreflexive.

where the `Begin_Turn` and `End_Turn` mark two limits of the time interval within which the concept is valid. Given the linear ordering of game turns `TSc` will clearly collect all these turns that are placed between `Begin_Turn` and `End_Turn`, including the two. In the next step we can define a time scope restriction which is imposed on any concept when necessary:

(TScR) `(applicable_from some (time:before some TSc))`
`and (applicable_to some (time:after some TSc))`

An individual falls under a concept restricted by (TScR) provided that the intersection of its applicability interval and the concept's time scope is not empty. Some possible cases are depicted below. All instances under the axis would satisfy the (TScR) of the concept.



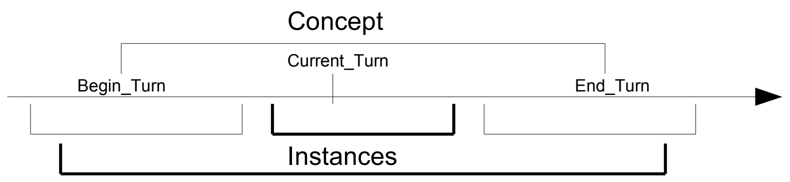
Finally, we have employed an auxiliary individual called `Current_Turn`, which plays a focusing role in the representation. The goal is to allow the reasoner for classifying only currently applicable instances and only under those concepts that represent the valid knowledge of the current turn. In this way the represented universe of entities indeed reflects the current state of affairs from ontological, as well as epistemological perspective. Depending on which game turn we want to investigate, we assert it as a value of the `owl:sameAs` property of `Current_Turn`, thus propagating this change to all restrictions in the ontology. Formally, focusing is obtained by augmenting the time scope restrictions with the `Current_Turn` individual. First, let's define a concept `TScC` as follows:

```
              TScC ≡ {Current_Turn} and TSc
                   ≡ {Current_Turn} and
(time:before has End_Turn) and (time:after has Begin_Turn)
```

`TScC` behaves like a function, which returns either the singleton containing the current turn, if the turn belongs to the time scope `TSc` of the concept, or the empty set if the current turn is outside of `TSc`. This property plays the key role in the revised form of time scope restriction, which is defined in a similar manner as before:

(TScCR) `(applicable_from some (time:before some TScC))`
`and (applicable_to some (time:after some TScC))`

An instance does not get classified under a concept restricted by (TScCR) unless the intersection of its applicability interval and the concept's scope is not empty, and moreover the current turn belongs to this intersection. This is the case only for two of the highlighted instances in the picture below.



Notice, that if `TScC` is empty, then also by reflexivity `(time:before \verb`some TScC)" and `(time:after some TScC))` cannot be satisfied by any instances of `Game_Turn`, and consequently no time stamped individual can satisfy (TScCR)[3].

How the representation functions should become clear in the following example. Let `Dynamic_Concept` be a concept changing over the time of a game. Let $\mathtt{C}_1$, $\mathtt{C}_2$ and $\mathtt{C}_3$ be its three consecutive variants, defined with the following necessary and sufficient conditions:

$$\begin{aligned}\mathtt{C}_1 &\equiv \mathtt{C\ and\ R}_1\\ \mathtt{C}_2 &\equiv \mathtt{C\ and\ R}_2\\ \mathtt{C}_3 &\equiv \mathtt{C\ and\ R}_3\end{aligned}$$

Let's further restrict the concepts with appropriate time scope conditions by instantiating `Begin_Turn` and `End_Turn` in the (TScCR) restrictions. For example:

$$\begin{aligned}\mathtt{C}_1 &\equiv \mathtt{C\ and\ R}_1\ \mathtt{and\ TScCR(1,3)}\\ \mathtt{C}_2 &\equiv \mathtt{C\ and\ R}_2\ \mathtt{and\ TScCR(4,7)}\\ \mathtt{C}_3 &\equiv \mathtt{C\ and\ R}_3\ \mathtt{and\ TScCR(8,10)}\end{aligned}$$

Now, let `I` be an instance satisfying restrictions `C`, $\mathtt{R}_2$, $\mathtt{R}_3$ and applicable from turn 2 to turn 9. Depending on which turn is set as the current one, the reasoner should infer the following types for `I`:

`Current_Turn`	Inferred type
1	none
3	none
5	C_2
9	C_3
10	none

[3]In some cases (TScCR) can be significantly simplified, saving some computational effort for the reasoner, while preserving full functionality. For all classes marked with the suffix `_Initial`, which represent the original variants of concepts (as given in the Initial Set of rules) it is possible to reduce (TScCR) only to its first conjunct `(applicable_from some (time:before some TScC)`. In case of all temporarily non-extensive entities (actions, proposals) it is already sufficient to use just `(applicable_from some TScC)`.

The final step in the representation, not fully accomplished within our model[4], is to define each concept that varies over time as the union of all its variants i.e.:

$$\texttt{Dynamic\_Concept} \equiv \texttt{C}_1 \sqcup \texttt{C}_2 \sqcup \ldots \sqcup \texttt{C}_n$$

where each variant $\texttt{C}_1$ to $\texttt{C}_n$ is restricted by means of its own necessary and sufficient conditions, including appropriate time scopes (as shown above). The task of classification of instances, which should be stored in some separate "bin" concept, has to be left entirely to the reasoner. Running the inference engine will enforce the denotation of the `Dynamic_Concept` to correspond to the current state of knowledge, or more precisely — to the currently valid meaning of the concept.

The rationale behind such a formalization is to enable easy cross-referencing between different parts of taxonomy in the epistemically dynamic setting. Consider again the initial example of this section concerning the change in the denotation of the concept `Voter`. By applying the proposed representation it is possible to refer to `Voter` in restrictions on other concepts, without having to update them every time the meaning of `Voter` changes. For instance, action `Vote` defined as:

```
Vote ≡ Game_Action and (action:actor some (action:Agent
                 some (plays some Voter)))
```

can remain unchanged even though the concept of `Voter` is revised in the course of a game. Whether a particular action of voting gets classified as a valid instance of the `Vote` concept depends on whether its actor plays indeed the role of a voter in the current understanding of this concept.

3.5 Remarks on Modeling Nomic Scenarios

For a working example of our model we have implemented a short scenario of a Nomic game to the OWL Nomic ontology (see the Appendix).

The scenario describes a 4-player game, comprising 6 turns, during which rules are being legally amended, transmuted, repealed and enacted, entailing respective changes in the terminology. By running the reasoner one can observe how different individuals get classified depending on their asserted properties on the one hand, and on the choice of the current turn on the other. For instance, in turn 2 the `_Initial` types (which are all still valid) are inferred for the proposal to amend rule 301, the voting for that proposal and its result, for the actual action of amending the rule, and further for roles of players and voters. Also the subset of rules in effect in turn 2 are extracted. Due to the amendment, a new variant of the concept `Voter` is introduced whose extension will therefore differ in the following turns.

[4]In fact only the concept of `Voter` is implemented in this way.

A few remarks listed below, which follow from our experience in dealing with the scenario, may serve as practical guidelines or useful reminders for similar tasks.

Reasoning over incomplete scenarios In the process of modeling an actual scenario it can be desirable to verify whether all the instances defined in a newly inputed turn receives the intended classification. Ideally, this step is supposed to serve for an automated legal assessment of proposed moves in a real-time game, so that any violations of game rules can be immediately detected. Verification is also recommended for practical reasons, since erroneous or incomplete assertions may affect the inferences over upcoming turns.

Although the game does not have to be finished at this stage yet, and many entities are presumably still valid, it is necessary to close all time stamps (for individuals) and time scopes (for concept variants) for the reasoning to succeed. Basically the closing is performed by asserting the currently modeled turn as the turn which all these entities are applicable to. In principle this might be a tedious procedure requiring numerous updates all over the knowledge base. To ease it we have introduced an auxiliary individual called `Last_Turn`. The individual has to be defined as `owl:sameAs` the last turn of the game which has been as far modeled. `Last_Turn` can be then used as the closing turn in time stamps and time scopes of all relevant entities. As a result, moving to the consecutive turn requires only a single update (viz. that of the `owl:sameAs` property of the `Last_Turn`) in order to extend applicability and validity of all the required individuals and concepts. The change is propagated to the whole ontology at once, saving the effort of updating every single individual and concept separately.

In practice we have consistently used `Last_Turn` as a closing turn every time a new (temporarily extensive) individual or concept variant is introduced in the game and when the end of its applicability interval is still not determined or simply goes beyond what has been as far modeled.

Concept revision When encountering a concept revision, one should follow the general modeling approach and define a new concept variant, placed as a sibling of the older one in the taxonomic tree. The old time scope has to be closed on the current turn, whereas the new one has to be opened starting from the following turn and ending in the `Last_Turn`.

Time stamping of rules Initially, whenever a rule change is applied, the rule that is the subject to change (except for the case of enactment) becomes applicable to the turn in which the change takes place, and a new, resulting rule is introduced (except for repeal) with the applicability interval starting from the following turn and ending in the `Last_Turn`.

4 Summary

The goal of the project was to explore the possibility of using LKIF-Core ontology for the task of modeling a piece of legislation. The object of modeling was the Initial Set of rules of Peter Suber's Nomic game — a game of an uncommon self-modifying character.

The resulting model, an OWL ontology, encompasses the normative knowledge conveyed by relatively small, but as we believe, the most essential and representative subset of Nomic's rules. The focus is predominantly on concepts necessary for expressing rule changes, i.e. actual actions of conducting a change, along with all required prerequisites, such as rules, players, proposals, actions of voting on proposals, etc. Within that scope, though quite limited, Nomic seems to reveal the most affinity with some typical legislative acts, and thus may serve as a simple but good example for modeling in LKIF-Core.

Due to the simplistic character of Nomic our experience of using LKIF-Core was inevitably restricted, and so are our conclusions following from it. In fact, there was no need to use the vast majority of LKIF concepts or relations, or even entire modules such as: `top:Occurence`, `expression:Medium`, `top:Physical_Concept` or `Modification`. Nevertheless, to the extent that legal terminology was indeed employed, LKIF-core turned out to be a very useful tool for the task, both as a ready upper-level taxonomical structure, allowing for faster and easier organization of terminological knowledge extracted from Nomic, but also as a heuristic guide, providing valuable hints on what concepts and relations should be searched for in the text of legislation. Especially the second role, which is not perhaps the primary objective of LKIF, should not be underestimated in cases were ontology developers differ in or simply lack sufficient legal background.

What we have found as a main limitation for the task (and as we think in principle for any task of modeling legislation) is the trade-off between fidelity and complexity of the representation. Building a well-grained OWL model of a legislation that would preserve the intended interpretation of natural language expressions quickly leads to blow-up in the number of used concepts and in the size and complexity of concept restrictions. As a consequence, human readability of the representation is severely affected and, what is more serious, the model becomes computationally intractable. Hence, the ideal of establishing a one to one correspondence between the law and its formal model based on LKIF-Core, and generally on OWL technology — a clear prerequisite for providing extensive and reliable automated legal assessment services — seems in practice to be a hardly achievable goal. LKIF-Core in itself is already a relatively complex structure, which considerably slows down the reasoning process, being a practical obstacle in development of the model. For that reason we left out some of unnecessary modules and branches of LKIF taxonomy, embedding the model in the subset of LKIF-Core, in the report referred to as LKIF-Skeleton. Selecting a subset of LKIF instead of the entire ontology as the basis for modeling, though always project-oriented and requiring some caution, might be in general suggested as a convenient strategy for effective usage of LKIF-Core.

Apart from representing the terminological knowledge of the initial state of game, we have also proposed a methodology of handling the changes in knowledge that can take place during the game. Our approach of representing the temporal aspects of Nomic allows for keeping track of all epistemic states that characterize consecutive turns of the game. By employing the focusing role of the auxiliary `Current_Turn` individual, we are able to reason over any selected turn of the game, verifying whether the game was played according to rules (and so according to terminological knowledge) which were in effect in that turn. In this sense it is possible, to some limited extent, to use the model for automated legal assessment of moves in the Nomic game.

Finally, to relate our work to another research on formalizing Nomic we shall briefly refer to the paper of G. Vreeswijk, [2]. The perspective on Nomic taken in our project is quite different from that considered by Vreeswijk. Whereas Vreeswijk focuses on the analysis of the criteria and general dynamics of allowing/forcing shifts between following epistemic states of the game, our major concern is to actually model the (normative) knowledge characterizing these states. Unlike Vreeswijk we have been not interested in grasping formal aspects of the game's rationality, but rather on representing deontic possibilities that are present on each stage of the game. An interesting overlap, however, especially from Vreeswijk perspective, would be to examine how particular changes affect the space of deontic possibilities, leading the game to stable or unstable states. In one extreme case a game might reach the state of ultimate liberation (in our representation this would correspond to the situation when there are no property restrictions on currently valid concepts) in the ether the "dead end" where nothing is allowed anymore (when currently valid concepts are inconsistent).

References

[1] Boer, Alexander (Eds), *Deliverable 1.4, OWL Ontology of Basic Legal Concepts (LKIF-Core)*, ESTRELLA Project, 2007.

[2] Vreeswijk, Gerard A. W., "Formalizing Nomic: working on a theory of communication with modifiable rules of procedure" in: *Technical report CS 95-02*, Vakgroep Informatica (FdAW), Rijksuniversiteit Limburg, Maastricht, The Netherlands. Presented at the Fourth International Symposium on Cognitive Science, Donostia — San Sebastian, May 3-6, 1995.

Appendix

A Sample Scenario of the Nomic Game

There are 4 players: Abdo, Ji, Marc, Szymon. The following actions are taken in the respective turns of the game.

All information about scoring typed in italics is not actually handled by the model.

Turn 1: Player Marc proposes proposal 301, which states: transmute immutable rule 105 into a mutable rule. The proposal is unanimously accepted by all players. The transmuted rule receives number 301. *Marc scores 2 points.*

Turn 2: Player Ji proposes proposal 302, which states: amend rule 301 into: "A player is an eligible voter, if it is his turn or one of the two previous turns was his". The proposal is unanimously accepted by all players. The amended rule receives the number 302. *Ji scores 6 points.*

Turn 3: Player Szymon proposes proposal 303, which states: repeal rule 203. The proposal is unanimously accepted by all eligible voters. *Szymon scores 5 points.*

Turn 4: Player Abdo proposes proposal 304, which states: amend rule 208 into: "The winner is the player with the least points after two circuits of turns". The proposal is rejected. *Abdo loses 5 points.*

Turn 5: Player Marc proposes proposal 305, which states: enact a new rule: "The game is over with the beginning of the 6-th turn". The proposal is unanimously accepted by all eligible voters. The rule receives the number 305. *Marc scores 3 points.*

Turn 6: Player Ji, having scored maximum number of points, is the winner of the game.