10

A Family of Script Languages Licenses and Contracts — Incomplete Sketch[1]

Drs. Arimoto Yasuhito, Chen Xiaoyi and Xiang Jianwen were co-partners in this study

Caveat

This chapter is incomplete. Its basis, [62], is even more so. We leave a number of formal semantic function definitions undefined.

Summary

Classical digital rights license languages, [3, 9, 10, 72, 74–76, 113, 117, 141, 154, 166, 167, 177, 188, 189, 206, 207, 220] applied to the electronic "downloading", payment and rendering (playing) of artistic works (for example music, literature readings and movies). In this chapter we generalise such applications languages and we extend the concept of licensing to also cover work authorisation (work commitment and promises) in health care and in public government. The digital works for these two new application domains are patient medical records, public government documents (Sects. 10.2–10.4.3) and bus transport contracts (Sect. 10.6).

Digital rights licensing for artistic works seeks to safeguard against piracy and to ensure proper payments for the rights to render these works. Health care and public government license languages seek to ensure transparent and professional (accurate and timely) health care, respectively 'good governance'. Bus transport contract languages seeks to ensure timely and reliable bus services by an evolving set of transport companies.

Proper mathematical definition of licensing languages seeks to ensure smooth and correct computerised management of licenses and contracts.

In this chapter we shall motivate and exemplify four license languages, the pragmatics and syntax of four (Sects. 10.2–10.6) of them as well as the formal semantics of one of them (Sect. 10.6).

[1]This is an edited version of [62].
Section 10.6 (Pages 309–326) is new and authored only by Dines Bjørner.

10.1 Introduction

10.1.1 Computing Science cum Programming Methodology

- This chapter is not about [so-called Theoretical] Computer Science:
  - ⋆ The study & knowledge of concepts that can ∃ "inside" computers.
  - ⋆ Establishing computational models.
  - ⋆ Proving foundational lemmas.
- This chapter is about Computing Science
  - ⋆ The study & knowledge of how to construct "those" things !
  - ⋆ No proving foundational lemmas as in TCS.
  - ⋆ Instead establishing method principles, techniques and tools
  - ⋆ for formal specification and design calculi,
  - ⋆ and verifying, model checking and formally testing properties.

10.1.2 Caveats

This document constitutes a comprehended set of R&D development notes. It is not a report, let alone a JAIST report, and it is certainly not a publishable science and/or technology paper. This document is to serve as a basis for further work on the design, pragmatics, semantics and syntax of license languages, for upcoming work on understanding permissions and obligations, for upcoming work on studying possible understandings of license languages in terms of game theory, transaction costs, and possibly other issues such as technological feasibilities. This document is grossly incomplete.

We paraphrase the above. We do so since it has shown difficult for some to understand that this is not a paper anywhere close to submission for publications, let alone an internal JAIST report. To repeat: these are working notes. They are being "constantly" revised[2].

The formal semantics given "late" in the chapter (Sect. 10.6) is a standard, near "classical" way of (i) securing that the author of that formalisation has understood the design of the language. (ii) That CSP + RSL[3] definition can be used to write users reference manuals for constructing, issuing and acting upon licenses, and (ii) as a basis for implementing trustworthy interpreters for licenses and contracts and for license contract uses; that is for possibly provably correct implement a distributed license and contract management (monitoring and control) system.

If you are looking for "deeper" results, results that span any family of license languages adhering to the basic semantic principles developed in this document, then this is not the document to read. Well, you had better first read this document, or the reports and paper(s) that are planned to emanate

[2]Well, they have not been worked on since late 2006. I hope, during the summer of 2009, to be able to completely revise this chapter into a publishable paper.

[3]`CSP`: [137, 138, 218, 222], `RSL`: [31–33, 101, 104, 106]

from this document. Then you may have to do the research that may lead to generic results. It is to be expected from such theoretical computer science work that a mathematical notation — invented explicitly for the purpose — will then "redefine" a suitably commensurate (congruent) and perhaps "vastly" generic sub-language, and the desired generic results are then proved to hold of that special notation "semantics".

10.1.3 On Licenses

License:
a right or permission granted in accordance with law by a competent authority
to engage in some business or occupation,
to do some act, or to engage in some transaction
which but for such license would be unlawful

Merriam Webster Online [232]

The concepts of licenses and licensing express relations between (i) *actors* (licensors (the authority) and licensees), (ii) *entities* (artistic works, hospital patients, public administration, citizen documents) and bus transport contracts and (iii) *functions* (on entities), and as performed by actors. By issuing a license to a licensee, a licensor wishes to express and enforce certain permissions and obligations: which functions on which entities the licensee is allowed (is licensed, is permitted) to perform. In this chapter we shall consider four[4] kinds of entities: (i) digital recordings of artistic and intellectual nature: music, movies, readings ("audio books"), and the like, (ii) patients in a hospital as represented also by their patient medical records, (iii) documents related to public government, and (iv) busses, time tables and road nets (of a bus transport system).

The *permissions* and *obligations* issues are, (1) for the owner (agent) of some intellectual property to be paid (an *obligation*) by users when they perform *permitted* operations (rendering, copying, editing, sub-licensing) on their works; (2) for the patient to be professionally treated — by medical staff who are basically *obliged* to try to cure the patient; (3) for public administrators and citizens to enjoy good governance: transparency in law making (national parliaments and local prefectures and city councils), in law enforcement (i.e., the daily administration of laws), and law interpretation (the judiciary) — by agents who are basically *obliged* to produce certain documents while being *permitted* to consult (i.e., read, perhaps copy) other documents; and (4) for bus passengers to enjoy reliable bus schedules — offered by bus transport companies on contract to, say public transport authorities and on sub-contract to other such bus transport companies where these transport companies are *obliged* to honour a contracted schedule.

[4]During our 2006 study we only studied and in [62] we only reported on the license languages related to (i–iii) below. The bus transport contract language, related to (iv), emerged during late 2008.

10.1.4 What Kind of Science Is This?

It is experimental computing science: The study and knowledge of how to design and construct software that is right, i.e., correct, and the right software, i.e., what the user wants. To study methods for getting the right software is interesting. To study methods for getting the software right is interesting. Domain development helps us getting the right software. Deriving requirements from domain descriptions likewise. Designing software from such requirements helps us get the software right. Understanding a domain and then designing license languages from such an understanding is new. We claim that computer-supported management of properly designed license languages is a hallmark of the e-Society.

10.1.5 What Kind of Contributions?

The experimental nature of the project being reported on is as follows: Postulate four domains. Describe these informally and formally. Postulate the possibility of license languages (LLs) that somehow relate to activities of respective domains. Design these – experimentally. Try discover similarities and differences between the four LLs (LL_{DRM}, LL_{HHLL}, LL_{PALL}, CL_{BUS}). Formalise the common aspects: $\mathcal{C}_{\mathsf{LL}}$. Formalise the three LLs — while trying to "parameterise" the $\mathcal{C}_{\mathsf{LL}}$ to achieve LL_{DRM}, LL_{HHLL}, LL_{PALL}, CL_{BUS}. This investigation is bound to tell us something, we hope.

The above outlines our ultimate goals. In reality, this chapter brings us only part of the way towards such a goal. To do the study as outlined in this section we first need complete the formal semantics of all the four languages.

10.2 Pragmatics of Three License Languages

- *By* **pragmatics** *we understand the study and practice of the factors that govern our choice of language in social interaction and the effects of our choice on others.*

In this section we shall rough-sketch-describe pragmatic aspects of the three domains of (1) production, distribution and consumption of artistic works, (2) the hospitalisation of patient, i.e., hospital health care and (3) the handling of law-based document in public government. The emphasis is on the pragmatics of the terms, i.e., the language used in these three domains. We leave the discussion of the bussing contract language till Sect. 10.6.

10.2.1 The Performing Arts: Producers and Consumers

The intrinsic entities of the performing arts are the artistic works: drama or opera performances, music performances, readings of poems, short stories, novels, or jokes, movies, documentaries, newsreels, etc. We shall limit our span to the scope of electronic renditions of these artistic works: videos, CDs or other. In this paper we shall not touch upon the technical issues of "downloading"(whether "streaming" or copying, or other). That and other issues should be analysed in [245].

Operations on Digital Works

For a consumer to be able to enjoy these works that consumer must (normally first) usually "buy a ticket" to their performances. The consumer, i.e., the theatre, opera, concert, etc., "goer" (usually) cannot copy the performance (e.g., "tape it"), let alone edit such copies of performances. In the context of electronic, i.e., digital renditions of these performances the above "cannots" take on a new meaning. The consumer may copy digital recordings, may edit these, and may further pass on such copies or editions to others. To do so, while protecting the rights of the producers (owners, performers), the consumer requests permission to have the digital works transferred ("downloaded") from the owner/producer to the consumer, so that the consumer can render ("play") these works on own rendering devices (CD, DVD, etc., players), possibly can copy all or parts of them, then possibly can edit all or parts of the copies, and, finally, possibly can further license these "edited" versions to other consumers subject to payments to "original" licensor.

License Agreement and Obligation

To be able to obtain these permissions the user agrees with the wording of some license and pays for the rights to operate on the digital works.

The Artistic Electronic Works: Two Assumptions

Two, related assumptions underlie the pragmatics of the electronics of the artistic works. The first assumption is that the format, the electronic representation of the artistic works is proprietary, that is, that the producer still owns that format. Either the format is publicly known or it is not, that is, it is somehow "secret". In either case we "derive" the second assumption (from the fulfilment of the first). The second assumption is that the consumer is not allowed to, or cannot operate[5] on the works by own means (software, machines). The second assumption implies that acceptance of a license results in the consumer receiving software that supports the consumer in performing all operations on licensed works, their copies and edited versions: rendering, copying, editing and sub-licensing.

[5]render, copy and edit

Protection of the Artistic Electronic Works

The issue now is: how to protect the intellectual property (i.e., artistic) and financial (exploitation) rights of the owners of the possibly rendered, copied and edited works, both when, and when not further distributed.

10.2.2 Hospital Health Care: Patients and Medical Staff

Citizens go to hospitals in order to be treated for some calamity (disease or other), and by doing so these citizens become patients. At hospitals patients, in a sense, issue a request to be treated with the aim of full or partial restitution. This request is directed at medical staff, that is, the patient authorises medical staff to perform a set of actions upon the patient. One could claim, as we shall, that the patient issues a license.

Hospital Health Care: Patients and Patient Medical Records

So patients and their attendant patient medical records (PMRs) are the main entities, the “works” of this domain. We shall treat them synonymously: PMRs as surrogates for patients. Typical actions on patients — and hence on PMRs — involve admitting patients, interviewing patients, analysing patients, diagnosing patients, planning treatment for patients, actually treating patients, and, under normal circumstance, to finally release patients.

Hospital Health Care: Medical Staff

Medical staff may request (‘refer’ to) other medical staff to perform some of these actions. One can conceive of describing action sequences (and ‘referrals’) in the form of hospitalisation (not treatment) plans. We shall call such scripts for licenses.

Professional Health Care

The issue is now, given that we record these licenses, their being issued and being honoured, whether the handling of patients at hospitals follow, or does not follow properly issued licenses.

10.2.3 Public Government and the Citizens

The Three Branches of Government

By public government we shall, following Charles de Secondat, baron de Montesquieu (1689–1755)[6], understand a composition of three powers: the law-making (legislative), the law-enforcing and the law-interpreting parts of public

[6] *De l'esprit des lois* (*The Spirit of the Laws*), published 1748

government. Typically national parliament and local (province and city) councils are part of law-making government. Law-enforcing government is called the executive (the administration). And law-interpreting government is called the judiciary [system] (including lawyers etc.).

Documents

A crucial means of expressing public administration is through *documents*.[7] We shall therefore provide a brief domain analysis of a concept of documents. (This document domain description also applies to patient medical records and, by some "light" interpretation, also to artistic works — insofar as they also are documents.)

Documents are *created*, *edited* and *read*; and documents can be *copied*, *distributed*, the subject of *calculations* (interpretations) and be *shared* and *shredded*.

Document Attributes

With documents one can associate, as attributes of documents, the *actors* who created, edited, read, copied, distributed (and to whom distributed),shared, performed calculations and shredded documents.

With these operations on documents, and hence as attributes of documents one can, again conceptually, associate the *location* and *time* of these operations.

Actor Attributes and Licenses

With actors (whether agents of public government or citizens) one can associate the *authority* (i.e., the *rights*) these actors have with respect to performing actions on documents. We now intend to express these *authorisations as licenses*.

Document Tracing

An issue of public government is whether citizens and agents of public government act in accordance with the laws — with actions and laws reflected in documents such that the action documents enables a trace from the actions to the laws "governing" these actions.

We shall therefore assume that every document can be traced back to its law-origin as well as to all the documents any one document-creation or -editing was based on.

[7]Documents are, for the case of public government to be the "equivalent" of artistic works.

10.3 The Semantic Intents of Licensor and Licensee Actions

10.3.1 Overview

There are two parties to a license: the *licensor* and the *licensee*. And there is a common agreement concerning a shared "item" between them, namely: the *license* and the *work item:* the artistic work, the patient, the document.

The licensor gives the licensee permission, or mandates the licensee to be obligated to perform certain actions on designated "items".

The licensee performs, or does not perform permitted and/or obligated actions

And the licensee may perform actions not permitted or not obligated.

The license shall serve to ensure that only permitted actions are carried out, and to ensure that all obligated actions are carried out.

Breach of the above, that is, breach of the contracted license may or shall result in revocation of the license.

10.3.2 Licenses and Actions

Licenses

Conceptually a licensor o (for owner) may issue a license named ℓ to licensee u (for user) to perform some actions. The license may syntactically appear as follows:

ℓ : **licensor** o **licenses licensee** u **to**
  perform actions {a1,a2,...,an} **on work item** w

Actions

And, conceptually, licensee (u) may perform actions which can be expressed as follows:

ℓ:a(w); ℓ:a′(w); ...; ℓ:a″(w); ...; ℓ:a‴(w)

These actions $(a, a', ..., a'', ..., a''')$ may be in the set {a1,a2,...,an}, mentioned in the license, or they may not be in that set. In the latter case we have a breach of license ℓ.

Any one licensee may have licensed several licenses $\ell_1, \ell_2, \ldots, \ell_n$. And such a licensee may, in an interleaved fashion, perform actions referring to different licenses:

$$\ell_i : a_i(w); \ell_j : a'_j(w); ...; \ell_k : a''_k(w); ...; \ell_n : a'''_n(w)$$

Two Languages

Thus there is *the language of licenses* and *the language of actions*.

We advise the reader to take note of the distinction between the permitted or obligated actions enumerated in a license and the license name labelled actions actually requested by a licensee.

10.3.3 Sub-licensing, Scheme I

A licensee u may wish to delegate some of the work associated with performing some licensed actions to a colleague (or customer). If, for example the license originally stated:

ℓ : **licensor** o **licenses licensee** u
    to perform actions {a1,a2,...,an} **on work item** w

the licensee (u) may wish a colleague u' to perform a subset of the actions, for example

{ai,aj,...,ak} $\subseteq$ {a1,a2,...,an}

Therefore u would like if the above license

ℓ : **licensor** o **licenses licensee** u
    to perform actions {a1,a2,...,an} **on work item** w

instead was formulated as:

ℓ : **licensor** o **licenses licensee** u
    to perform actions {a1,a2,...,an} **on work item** w
    allowing sub-licensing of actions {ai,aj,...,ak}

where

{ai,aj,...,ak} $\subseteq$ {a1,a2,...,an}

Now licensee u can perform the action

ℓ : **license actions** {a$'$,a$''$,a$'''$} **to** u$'$

The above is an action designator. Its practical meaning is that a license is issued by u:

$\eta(\ell$,u,t): **licensor** u **licenses licensee** u$'$
    to perform actions {a$'$,a$''$,a$'''$} **on work item** w

The above license can be easily "assembled" from the action including the action named license: the context determines who (namely u) is issuing the license, and who or which is the work item. η is a function which applies to license name, agent identifications and time and yields unique new license names.

10.3.4 Sub-licensing, Scheme II

The subset relation

{ai,aj,...,ak} ⊆ {a1,a2,...,an}

mentioned in the sub-licensing part of license

ℓ : **licensor** o **licenses licensee** u
  to perform actions {a1,a2,...,an} **on work item** w
  allowing sub-licensing of actions {ai,aj,...,ak}

may be omitted. In fact one could relax the relation completely and allow any actions {ai,aj,...,ak} whether in {a1,a2,...,an} or not ! That is, the original licensor may mandate that a licensee allow a sub-licensee to perform operations that the licensee is not allowed to perform. Examples are: a licensee may break the shrink-wrap around some licensed software package — an action which may not be performed by the licensor; a medical nurse (i.e., a licensee) may perform actions on patients not allowed performed by the licensor (say, a medical doctor); and a civil servant (say, an archivist) may copy, distribute or shred documents, actions that may not be allowed by the licensor (i.e., the manager of that civil servant), while that civil servant (the archivist) is not allowed to create or read documents.

10.3.5 Multiple Licenses

Consider the following scenario: A licensee u is performing actions a_p, a_q, $\ldots$, a_r, on work item ω, and has licensed u' to perform actions $a_i, a_j, \ldots, a_k$, also on work item ω. The action whereby u licenses u' occurs at some time. At that time u has performed none or some of the actions $a_p, a_q, \ldots, a_r$ (on work item ω), but maybe not all. What is going to happen? Can u and u' go on, in parallel, performing actions on the same work item (ω) ? Our decision is yes, and they can do so in an interleaved manner, not concurrently but alternating, i.e., not accessing the same work item simultaneously.

10.4 Syntax and Informal Semantics

We distinguish between the pragmatics, the semantics and the syntax of languages. Leading textbooks on (formal) semantics of programming languages are [82, 114, 215, 221, 236, 241].

We have already covered the concept of pragmatics and illustrated its application to some issues of license language design.

We shall now illustrate the use of syntax and semantics in license language design.

- *By* **syntax** *we mean (i) the ways in which words are arranged to show meaning (cf. semantics) within and between sentences, and (ii) rules for forming syntactically correct sentences.*
- *By* **semantics** *we mean the study and knowledge [incl. specification] of meaning in language [79].*
- By **informal** *semantics we mean a semantics which is expressed in concise natural language, for example, as here, English.*

10.4.1 General Artistic License Language

We refer to the abstract syntax formalised below (that is, formulas 0.–14.). The work on the specific form of the syntax has been facilitated by the work reported by Xiang JenWen [245].[8]

Licenses and Actions

The syntax has two parts. One for licenses being issued by licensors. And one for the actions that licensees may wish to perform.

Licenses

Syntax

We first present an abstract syntax of the language of licenses, then we annotate this abstract syntax, and finally we present an informal semantics of that language of licenses.

type
0. Ln, Nm, W, S, V
1. L = Ln × Lic
2. Lic == mkLic(licensor:Nm,licensee:Nm,work:W,cmds:Cmd-**set**)
3. Cmd == Rndr | Copy | Edit | RdMe | SuLi
4. Rndr = mkRndr(vw:(V|″`work`″),sl:S^*)
5. Copy = mkCopy(fvw:(V|″`work`″),sl:S^*,tv:V)
6. Edit = mkEdit(fvw:(V|″`work`″),sl:S^*,tv:V)
7. RdMe = ″`readme`″
8. SuLi = mkSuLi(cs:Cmd-**set**,work:V)

[8] As this work, [245], has yet to be completed the syntax and annotations given here may change.

Syntax Annotations

0: Syntax Sorts (0.) Licenses are given names, ln:Ln, so are actors (owners, licensors, and users, licensees), nn:Nm. By w:W we mean a (net) reference to (a name of) the downloaded possibly segmented artistic work being licensed, where segments are named (s:S), that is, s:S is a selector to either a segment of a downloaded work or to a segment of a copied and or and edited work.

(1.) Every license (lic:Lic) has a unique name (ln:Ln).

(2.) A license (lic:Lic) contains four parts: the name of the licensor, the name of the licensee, a reference to (the name of) the work, a set of commands (that may be permitted to be performed on the work).

(3.) A command is either a render, a copy or an edit or a readme command, or a sub-licensing (sub-license) command.

(4.–6.) The render, copy and edit commands are each "decorated" with an ordered list of selectors (i.e., selector names) and a (work) variable name. The license command

copy $\langle$s1,s2,s7$\rangle$ v

means that the licensed work, ω, may have its sections s_1, s_2 and s_7 copied, in that sequence, into a new variable named v, Other copy commands may specify other sequences. Similarly for render and edit commands.

(7.) The "readme" license command, in a license, ln, referring, by means of w, to work ω, somehow displays a graphical/textual "image" of, that is, information about ω. We do not here bother to detail what kind of information may be so displayed. But you may think of the following display information names of artistic work,artists, authors, etc., names and details about licensed commands, a table of fees for performing respective licensed commands, etcetera.

(8.) The license command

license cmd1,cmd2,...,cmdn **on work** v
mkSuLi({cmd1,cmd2,...,cmdn},v)

means that the licensee is allowed to act as a licensor, to name sub-licensees (that is, licensees) freely, to select only a (possibly full) subset of the sub-licensed commands (that are listed) for the sub-licensee to enjoy. The license need thus not mention the name(s) of the possible sub-licensees. But one could design a license language, i.e., modify the present one to reflect such constraints. The license also do not mention the payment fee component. As we shall see under licensor actions such a function will eventually be inserted.

Informal Semantics

A license licenses the licensee to render, copy, edit and license (possibly the results of editing) any selection of downloaded works. In any order — but see below — and any number of times. For every time any of these operations take place payment according to the payment function occurs (that can be inspected by means of the **read license** command). The user can render the downloaded work and can render copies of the work as well as edited versions of these. Edited versions are given own names. Editing is always of copied versions. Copying is either of downloaded or of copied or edited versions. This does not transpire from the license syntax but is expressed by the licensee, see below, and can be checked and duly paid for according to the payment function.

The payment function is considered a partial function of the selections of the work being licensed.

Please recall that licensed works are proprietary. Either the work format is known, or it is not supposed to be known, In any case, the rendering, editing, copying and the license-“assembling” (see next section) functions are part of the license and the licensed work and are also assumed to be proprietary. Thus the licensee is not allowed to and may not be able to use own software for rendering, editing, copying and license assemblage.

Licenses specify sets of permitted actions. Licenses do not normally mandate specific sequences of actions. Of course, the licensee, assumed to be an un-cloned human, can only perform one action at a time. So licensed actions are carried out sequentially. The order is not prescribed, but is decided upon by the licensee. Of course, some actions must precede other actions. Licensees must copy before they can edit, and they usually must edit some copied work before they can sub-license it. But the latter is strictly speaking not necessary.

Actions

Syntax

type
9. V
10. Act = Ln × (Rndr|Copy|Edit|License)
11. Rndr == mkR(sel:S*,wrk:(W|V))
12. Copy == mkC(sel:S*,wrk:(W|V),into:V)
13. Edit == mkE(wrks:V*,into:V)
14. License == mkL(ln:Ln,licensee:Nm,wrk:V,cmds:Cmd-**set**,fees:PF)

Annotations and Informal Semantics:

9: Variables By V we mean the name of a variable in the users own storage into which downloaded works can be copied (now becoming a local work. The variables are not declared. They become defined when the licensee names them in a copy command. They can be overwritten. No type system is suggested.

10: Actions Every action of a licensee is tagged by the name of a relevant license. If the action is not authorised by the named license then it is rejected. Render and copy actions mention a specific sequence of selectors. If this sequence is not an allowed (a licensed) one, then the action is rejected. (Notice that the license may authorise a specific action, a with different sets of sequences of selectors — thus allowing for a variety of possibilities as well as constraints.)

11: Render The licensee, having now received a license, can render selections of the licensed work, or of copied and/or edited versions of the licensed work. No reference is made to the payment function. When rendering the semantics is that this function is invoked and duly applied. That is, render payments are automatically made: subtracted from the licensees account and forwarded to the licensor.

12: Copy The licensee can copy selections of the licensed work, or of previously copied and/or edited versions of the licensed work. The licensee identifies a name for the local storage file where the copy will be kept. No reference is made to the payment function. When copying the semantics is that this function is invoked and duly applied. That is, copy payments are automatically made: subtracted from the licensees account and forwarded to the licensor.

13: Edit The licensee can edit selections of the licensed work, or of copied and/or previously edited versions of the licensed work. The licensee identifies a name for the local storage file where the new edited version will be kept. The result of editing is a new work. No reference is made to the payment function. When copying the semantics is that this function is invoked and duly applied. That is, copy payments are automatically made: subtracted from the licensees account and forwarded to the licensor. Although no reference is made to any edit functions these are made available to the licensee when invoking the edit command. You may think of these edit functions being downloaded at the time of downloading the license. Other than this we need not further specify the editing functions. Same remarks apply to the above copying functions.

14: Sub-Licensing The licensee can further sub-license copied and/or edited work. The licensee must give the license being assembled a unique name. And the licensee must choose to whom to license this work. A sub-license, like does a license, authorises which actions can be performed, and then with which one of a set of alternative selection sequences. No payment function is explicitly mentioned. It is to be semi-automatically derived (from the originally licensed payment fee function and the licensee's payment demands) by means of functionalities provided as part of the licensed payment fee function.

The sub-license command information is thus compiled (assembled) into a license of the form given in (1.–3.). The licensee becomes the licensor and the recipient of the new, the sub-license, become the new licensee. The assemblage refers to the context of the action. That context knows who, the licensor, is issuing the sub-license. From the license label of the command it is known whether the sub-license actions are a subset of those for which sub-licensing has been permitted.

10.4.2 Hospital Health Care License Language

We refer to the abstract syntax formalised below (that is, formulas 1.–5.). The work on the specific form of the syntax has been facilitated by the work reported in [8].[9]

Licenses and Actions

The syntax has two parts. One for licenses being issued by licensors. And one for the actions that licensees may wish to perform.

A Notion of License Execution State

In the context of the Artistic License Language licensees could basically perform licensed actions in any sequence and as often as they so desired. There were, of course, some obvious constraints. Operations on local works could not be done before these had been created — say by copying. Editing could only be done on local works and hence required a prior action of, for example, copying a licensed work. In the context of hospital health care most of the actions can only be performed if the patient has reached a suitable state in the hospitalisation. We refer to Fig. 10.1 on the following page for an idealised hospitalisation plan.

We therefore suggest to join to the licensed commands an indicator which prescribe the (set of) state(s) of the hospitalisation plan in which the command action may be performed.

Two or more medical staff may now be licensed to perform different (or even same !) actions in same or different states. If licensed to perform same action(s) in same state(s) — well that may be "bad license programming" if and only if it is bad medical practice ! One cannot design a language and prevent it being misused!

Licenses

type
0. Ln, Mn, Pn
1. License = Ln × Lic
2. Lic == mkLic(staff1:Mn,mandate:ML,pat:Pn)
3. ML == mkML(staff2:Mn,to_perform_acts:CoL-**set**)
4 CoL = Cmd | ML | Alt

[9] As this work, [8], has yet to be completed the syntax and annotations given here may change.

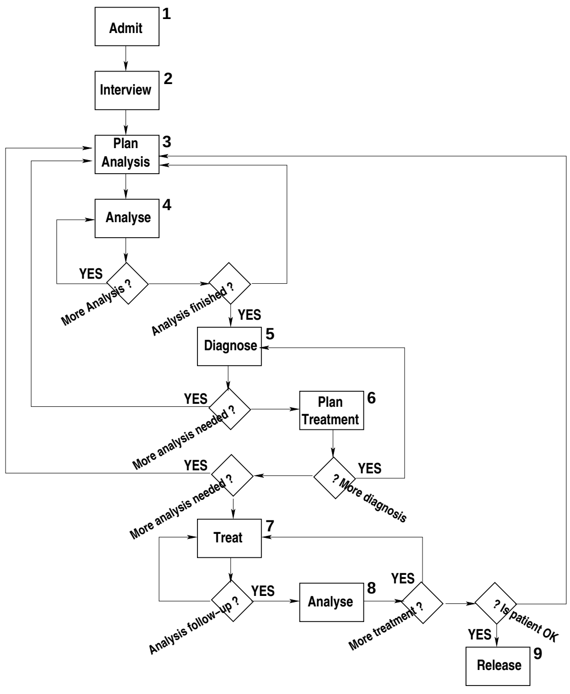

Fig. 10.1. An example hospitalisation plan. States: {1,2,3,4,5,6,7,8,9}

5. Cmd == mkCmd(σs:Σ-**set**,stmt:Stmt)
6 Alt == mkAlt(cmds:Cmd-**set**)
7. Stmt = **admit** | **interview** | **plan-analysis** | **do-analysis**
 | **diagnose** | **plan-treatment** | **treat** | **transfer** | **release**

The above syntax is correct RSL [31–33, 101, 104, 106]. But it is decorated! The subtypes {|**boldface keyword**|} are inserted for readability.

Syntax Annotations

(0.) Licenses, medical staff and patients have names.

(1.) Licenses further consist of license bodies (Lic).

(2.) A license body names the licensee (Mn), the patient (Pn), and,

(3.) through the "mandated" licence part (ML), it names the licensor (Mn) and which set of commands (C) or (o) implicit licenses (L, for CoL) the licensor is mandated to issue.

(4.) An explicit command or licensing (CoL) is either a command (Cmd), or a sub-license (ML) or an alternative.

(5.) A command (Cmd) is a state-labelled statement.

(3.) A sub-license just states the command set that the sub-license licenses. As for the Artistic License Language the licensee chooses an appropriate subset of commands. The context "inherits" the name of the patient. But the sub-licensee is explicitly mandated in the license!

(6.) An alternative is also just a set of commands. The meaning is that either the licensee choose to perform the designated actions or, as for ML, but now freely choosing the sub-licensee, the licensee (now new licensor) chooses to confer actions to other staff.

(7.) A statement is either an admit, an interview, a plan analysis, an analysis, a diagnose, a plan treatment, a treatment, a transfer, or a release directive Information given in the patient medical report for the designated state inform medical staff as to the details of analysis, what to base a diagnosis on, of treatment, etc.

Actions

8. Action = Ln × Act
9. Act = Stmt | SubLic
10. SubLic = mkSubLic(sublicensee:Ln,license:ML)

Syntax Annotations

(8.) Each action actually attempted by a medical staff refers to the license, and hence the patient name.

(9.) Actions are either of an admit, an interview, a plan analysis, an analysis, a diagnose, a plan treatment, a treatment, a transfer, or a release actions. Each individual action is only allowed in a state σ if the action directive appears in the named license and the patient (medical record) designates state σ.

(10.) Or an action can be a sub-licensing action. Either the sub-licensing action that the licensee is attempting is explicitly mandated by the license (4. ML), or is an alternative one thus implicitly mandated (6.). The full sub-license, as defined in (1.–3.) is compiled from contextual information.

Informal Semantics

An informal, rough-sketch semantics (here abbreviated) would state that a prescribed action is only performed if the patient, cum patient medical record is in an appropriate state; and that the patient is being treated according to the action performed; that records of this treatment and its (partially) analysed outcome is introduced into the patient medical record. The next state of the patient, cum patient medical record, depends on the outcome of the treatment[10]; and hence the patient medical record carries with it, i.e., embodies a, or the, hospitalisation plan in effect for the patient, and a reference to the current state of the patient.

[10]Cf. the diamond-shaped decision boxes in Fig. 10.1 on the preceding page.

10.4.3 Public Administration License Language

We refer to the abstract syntax formalised below (that is, formulas 1.–15.). The work on the specific form of the syntax has been facilitated by the work reported in [26, 46, 73].[11]

Licenses and Actions

The syntax has two parts. One for licenses being issued by licensors. And one for the actions that licensees may wish to perform.

Licenses

License Classes

```
type
0. Ln, An, Cfn
1. L          == Grant | Extend | Restrict | Withdraw
2. Grant      == mkG(license:Ln,licensor:An,granted_ops:Op-set,licensee:An)
3. Extend     ==  mkE(licensor:An,licensee:An,license:Ln,with_ops:Op-set)
4. Restrict   == mkR(licensor:An,licensee:An,license:Ln,to_ops:Op-set)
5. Withdraw == mkW(licensor:An,licensee:An,license:Ln)
6. Op         == Crea|Edit|Read|Copy|Licn|Shar|Rvok|Rlea|Rtur|Calc|Shrd
```

Licensed Operations

```
type
7.  Dn, DCn, UDI
8.  Crea  == mkCr(dn:Dn,doc_class:DCn,based_on:UDI-set)
9.  Edit  == mkEd(doc:UDI,based_on:UDI-set)
10.  Read  == mkRd(doc:UDI)
11.  Copy  == mkCp(doc:UDI)
12. Licn  == mkLi(kind:LiTy)
13. LiTy  == grant | extend | restrict | withdraw
14.  Shar  == mkSh(doc:UDI,with:An-set)
15.  Rvok  == mkRv(doc:UDI,from:An-set)
16.  Rlea  == mkRl(dn:Dn)
17.  Rtur  == mkRt(dn:Dn)
18.  Calc  == mkCa(fcts:CFn-set,docs:UDI-set)
19.  Shrd  == mkSh(doc:UDI)
```

[11] As part this work, [73], has yet to be completed the syntax and annotations given here may change.

Syntax & Informal Semantics Annotations

(0.) The are names of licenses (Ln), actors (An), documents (UDI), document classes (DCn) and calculation functions (Cfn).

(1.) There are four kinds of licenses: granting, extending, restricting and withdrawing.

(2.) Actors (licensors) grant licenses to other actors (licensees). An actor is constrained to always grant distinctly named licenses. No two actors grant identically named licenses.[12] A set of operations on (named) documents are granted.

(3.–5.) Actors who have issued named licenses may extend, restrict or withdraw the license rights (wrt. operations, or fully).

(6.) There are nine kinds of operation authorisations. Some of the next explications also explain parts of some of the corresponding actions (see (16.–24.).

(7.) There are names of documents (Dn), names of classes of documents (DCn), and there are unique document identifiers (UDI).

(8.) Creation results in an initially void document which is

not necessarily uniquely named (dn:Dn) (but that name is uniquely associated with the unique document identifier created when the document is created[13]) typed by a document class name (dcn:DCn) and possibly based on one or more identified documents (over which the licensee (at least) has reading rights). We can presently omit consideration of the document class concept. "based on" means that the initially void document contains references to those (zero, one or more) documents.[14] The "based on" documents are moved from licensor to licensee.

(9.) Editing a document may be based on "inspiration" from, that is, with reference to a number of other documents (over which the licensee (at least) has reading rights). What this "be based on" means is simply that the edited document contains those references. (They can therefore be traced.) The "based on" documents are moved from licensor to licensee — if not already so moved as the result of the specification of other authorised actions.

(10.) Reading a document only changes its "having been read" status (etc.) — as per [26]. The read document, if not the result of a copy, is moved from licensor to licensee — if not already so moved as the result of the specification of other authorised actions.

(11.) Copying a document increases the document population by exactly one document. All previously existing documents remain unchanged except that the document which served as a master for the copy has been so marked. The copied document is like the master document except that the copied

[12]This constraint can be enforced by letting the actor name be part of the license name.

[13]— hence there is an assumption here that the create operation is invoked by the licensee exactly (or at most) once.

[14]They can therefore be traced (etc.) — as per [26].

document is marked to be a copy (etc.) — as per [26]. The master document, if not the result of a create or copy, is moved from licensor to licensee — if not already so moved as the result of the specification of other authorised actions.

(12.) A licensee can sub-license (sL) certain operations to be performed by other actors.

(13.) The granting, extending, restricting or withdrawing permissions, cannot name a license (the user has to do that), do not need to refer to the licensor (the licensee issuing the sub-license), and leaves it open to the licensor to freely choose a licensee. One could, instead, for example, constrain the licensor to choose from a certain class of actors. The licensor (the licensee issuing the sub-license) must choose a unique license name.

(14.) A document can be shared between two or more actors. One of these is the licensee, the others are implicitly given read authorisations. (One could think of extending, instead the licensing actions with a **shared** attribute.) The shared document, if not the result of a create and edit or copy, is moved from licensor to licensee — if not already so moved as the result of the specification of other authorised actions. Sharing a document does not move nor copy it.

(15.) Sharing documents can be revoked. That is, the reading rights are removed.

(16.) The release operation: if a licensor has authorised a licensee to create a document (and that document, when created got the unique document identifier udi:UDI) then that licensee can release the created, and possibly edited document (by that identification) to the licensor, say, for comments. The licensor thus obtains the master copy.

(17.) The return operation: if a licensor has authorised a licensee to create a document (and that document, when created got the unique document identifier udi:UDI) then that licensee can return the created, and possibly edited document (by that identification) to the licensor — "for good"! The licensee relinquishes all control over that document.

(18.) Two or more documents can be subjected to any one of a set of permitted calculation functions. These documents, if not the result of a creates and edits or copies, are moved from licensor to licensee — if not already so moved as the result of the specification of other authorised actions. Observe that there can be many calculation permissions, over overlapping documents and functions.

(19.) A document can be shredded. It seems pointless to shred a document if that was the only right granted wrt. document.

Actions

20. Action = Ln × Clause
21. Clause = Cre | Edt | Rea | Cop | Lic | Sha | Rvk | Rel | Ret | Cal | Shr
22. Cre == mkCre(dcn:DCn,based_on_docs:UID-**set**)
23. Edt == mkEdt(uid:UID,based_on_docs:UID-**set**)
24. Rea == mkRea(uid:UID)

```
25. Cop == mkCop(uid:UID)
26. Lic == mkLic(license:L)
27. Sha == mkSha(uid:UID,with:An-set)
28. Rvk == mkRvk(uid:UID,from:An-set)
29. Rel == mkRel(dn:Dn,uid:UID)
30. Ret == mkRet(dn:Dn,uid:UID)
31. Cal == mkCal(fct:Cfn,over_docs:UID-set)
32. Shr == mkShr(uid:UID)
```

Preliminary Remarks

A clause elaborates to a state change and usually some value. The value yielded by elaboration of the above create, copy, and calculation clauses are unique document identifiers. These are chosen by the "system".

Syntax & Informal Semantics Annotations

(20.) Actions are tagged by the name of the license with respect to which their authorisation and document names has to be checked. No action can be performed by a licensee unless it is so authorised by the named license, both as concerns the operation (create, edit, read, copy, license, share, revoke, calculate and shred) and the documents actually named in the action. They must have been mentioned in the license, or, created or copies of downloaded (and possibly edited) documents or copies of these — in which cases operations are inherited.

(21.) Actions clauses are either create, edit, read, copy, sub-license, share, revoke, release, return, calculate or shred.

(22.) A licensee may create documents if so licensed — and obtains all operation authorisations to this document.

(23.) A licensee may edit "downloaded" (edited and/or copied) or created documents.

(24.) A licensee may read "downloaded" (edited and/or copied) or created and edited documents.

(25.) A licensee may (conditionally) copy "downloaded" (edited and/or copied) or created and edited documents. The licensee decides which name to give the new document, i.e., the copy. All rights of the master are inherited to the copy.

(26.) A licensee may issue licenses of the kind permitted. The licensee decides whether to do so or not. The licensee decides to whom, over which, if any, documents, and for which operations. The licensee looks after a proper ordering of licensing commands: first grant, then sequences of zero, one or more either extensions or restrictions, and finally, perhaps, a withdrawal.

(27.) A "downloaded" (possibly edited or copied) document may (conditionally) be shared with one or more other actors. Sharing, in a digital world, for example, means that any edits done after the opening of the sharing session, can be read by all so-granted other actors.

(28.) Sharing may (conditionally) be revoked, partially or fully, that is, wrt. original "sharers".

(29.) A document may be released. It means that the licensor who originally requested a document (named dn:Dn) to be created now is being able to see the results — and is expected to comment on this document and eventually to re-license the licensee to further work.

(30.) A document may be returned. It means that the licensor who originally requested a document (named dn:Dn) to be created is now given back the full control over this document. The licensee will no longer operate on it.

(31.) A license may (conditionally) apply any of a licensed set of calculation functions to "downloaded" (edited, copied, etc.) documents, or can (unconditionally) apply any of a licensed set of calculation functions to created (etc.) documents. The result of a calculation is a document. The licensee obtains all operation authorisations to this document (— as for created documents).

(32.) A license may (conditionally) shred a "downloaded" (etc.) document.

10.4.4 Discussion

Comparisons

We have "designed", or rather proposed three different kinds of license languages. In which ways are they "similar", and in which ways are they "different"? Proper answers to these questions can only be given after we have formalised these languages. The comparisons can be properly founded on comparing the semantic values of these languages.

But before we embark on such formalisations we need some informal comparisons so as to guide our formalisations. So we shall attempt such analysis now with the understanding that it is only a temporary one.

Differences

Work Items The work items of the artistic license language(s) are essentially "kept" by the licensor. The work items of the hospital health care license language(s) are fixed and, for a large set of licenses there is one work item, the patient which is shared between many licensors and licenses. The work items of the public administration license language(s) — namely document — are distributed to or created and copied by licenses and may possibly be shared.

Operations The operations of the artistic license language(s) are are essentially "kept" by the licensor. The operations of the hospital health care license language(s) are are essentially "kept" by the licensees (as reflected in their professional training and conduct). The operations of the public administration license language(s) are essentially "kept" by the licensees (as reflected in their professional training and conduct).

Permissions and Obligations Generally we can say that the modalities of the artistic license language(s) are essentially permissions with **payment** (as well as use of licensor functions) being an obligation; that the modalities of the hospital health care license language(s) are are essentially obligations; and, as well, that the modalities of the public administration license language(s) are essentially obligations We shall have more to say about permissions and obligations later (much later).

10.5 Formal Semantics

By formal semantics we understand a definition expressed in a formal language, that is, a language with a mathematical syntax, a mathematical semantics, and a consistent and relative complete proof system. We shall deploy the CSP [137, 138, 218, 222] Specification Language embedded in a Landin–like notation of **let** clauses[15]. We hope someone will complement our definition with a commensurate CafeOBJ [89, 90, 99, 100] definition.

10.5.1 A Model of Common Aspects

Actors: Behaviours and Processes

We see the system as a set of actors. An actor is either a licensor, or a licensee, or, usually, such as we have envisaged our license languages, both. To each actor we associate a behaviour — and we model actor behaviours as CSP processes. So the system is then modelled as a set of concurrent behaviours, that is, parallel (‖) CSP processes. Actors are uniquely identified (Aid).

System States

With each actor behaviour we associate a state ($\omega : \Omega$). "Globally" initial such state components are modelled as maps from actor identifiers to states. We shall later analyse these states into major components.

type
  Aid, Ω
  Ωs = Aid $\underset{m}{\rightarrow}$ Ω

System Processes

Actor processes communicate with one another over channels. There is a set of actor identifier indexed channels. Potential licensees request licenses. Licensors issue licenses in response to requests. Work items are communicated over these channels. As are payments and reports on use of licensed operations on

[15]— known since the very early 1960's

licensed work items. An actor is either pro-active, requesting licenses, sending payment or reports, or re-active: responding to license requests, sending work items. An actor non-deterministically ($\lceil\rceil$) alternates between these activities.

```
type
    M = Lic | Pay | Rpt | ...
channel
    {a[i]|i:Aid} (Aid×M)
value
    actor: j:Aid × Ω → in,out {a[i]|i:Aid•i≠j} Unit
    actor(j)(ω) ≡ let ω′ = pro_act(j)(ω)⌈⌉re_act(j)(ω) in actor(j)(ω′) end

    system: Ωs → Unit
    system(ωs) ≡ ‖ {actor(i)(ωs(i))|i:Aid}
```

Actor Processes

We have identified two kinds of actor processes: pro-active, during which the actor, by own initiative, (as a prospective licensee) may request a license from a prospective licensor, or, (as an actual licensee) as the result of performing licensed actions sends payments or reports (or other) to the licensor. and re-active, during which the actor, in response to requests (as a licensor) sends a requesting actor a license (whereby the requester becomes a licensee), or "downloads" (access to) requested works or functions. functions.

The Pro-active Actor Behaviour In the pro-active behaviour an actor, at will, i.e., non-deterministically internal choice ($\lceil\rceil$), decides to either request a license (rl) or to perform some action (op). In the former case the actor inquires (l_iq) an own state to find out from which licensor (aid) which kind of license requirements (l_rq) is to be requested. This licensor and these requirements are duly noted in the state. After sending the request the actor continues being an actor in the duly noted state. In the latter case (op) there may be many "next" actions to do (act). The actor inquires (a_iq) an own state to find out which action (designated by op_i) is "next". The actor them performs (act) the designated operation. It is here, for simplification assumed that all operation completions imply a "completion" message (a payment, a report, or other) to the operation licensing actor. So such a message is sent and the operation updated local state is yielded — whereby the pro-active actor "resumes" being an actor as from that state.

```
type
    M = Lic | Pay | Rpt | ...
channel
    {a[i]|i:Aid} (Aid×M)
value
    pro_act: j:Aid → Ω → in,out {a[i]|i:Aid•j≠i}  Ω
```

```
pro_act(j)(ω) ≡
    let what_to_do = rl ⌈⌉ op in
    case what_to_do of
        rl → let (k,l_rq,ω′)=iq_l_Ω(ω) in
                a(aid)!(j,l_rq);ω′ end
        op → let op_i=iq_a_Ω(ω) in
                let (k,m,ω′)=act(op_i)(ω) in
                a(k)!(j,m) ; ω′ end end
    end end
```

The Re-active Actor Behaviour In the re-active behaviour an actor (j), is willing to engage in communication with other actors. This is formalised by a non-deterministic external choice ($\lceil\rceil$) between either of a set ({...}) of (zero, or more) other actors (k:Aid\{j}) who are trying to contact the re-active actor. The communicated message reveals the identity (k) of the requesting, i.e., the pro-active actor,[16] The message, m, reveals what action (act(m)) the re-active actor is requested to perform. The actor does so/ This results in a reply message m′ and a state change. The reply message is sent to the requesting actor; and the re-active actor yields the requested action-updated state — whereby the re-active actor "resumes" being an actor as from that state.

```
type
    M = Lic | Pay | Rpt | ...
channel
    {a[i]|i:Aid} (Aid×M)
value
    re_act: j:Aid → Ω → in,out {a[i]|i:Aid•j≠i} Ω
    re_act(j)(ω)≡
        let (k,m)=⌈⌉{a(k)?|k:Aid} in
        let (m′,ω′)=act(m)(ω) in
        a(k)!(j,m′);ω′ end end
```

Functions

We first list (and "read") the signatures of the two auxiliary (*iq_l_Ω*, *iq_a_Ω*) and one elaboration (*act*) function assumed in the definition of the pro- and re-active actor processes. After that we discuss the former and suggest definitions of the latter.

[16] Do not get confused by the two k's on either side of the let clause. The left k is yielded by the (input) communication a(k)?. The defining scope of the right side k, as in a(k), is just the right-hand side of the left clause.

Auxiliary Function Signatures

The inquire license function (iq_l_Ω) inspects the actor's state to ("eureka") determine which most desirable licensor (Aid) offers which one kind of desired license requirements (License_Requirements). The inquire action function (iq_a_Ω) inspects the actor's state to (somewhat "eureka") determine which action is "next" in tine to be performed. That action is being designated (Action_Designator).

type
  License_Requirements,Action_Designation
value
  iq_l_Ω: $\Omega \rightarrow$ Aid $\times$ License_Requirements $\times\ \Omega$
  iq_a_Ω: $\Omega \rightarrow$ Action_Designator

By 'eureka'[17] is meant that the inquiry is internal non-deterministic that is, is not influenced by an outside, could have any one of very many outcomes, and can thus only be rather loosely defined.

Elaboration Function Signature

The action performing function (act) "finds" the designated operation in the current state, applies it in the current state, and yields ("read" backwards) a possibly new state ($\omega : \Omega$), a message (M) to be sent to the licensor (Aid) who authorised the operation and may need or which to have a payment, a report, or some such thing "back"!

type
  Action_Designation
value
  act: Action_Designation $\rightarrow \Omega \rightarrow$ Aid $\times$ M $\times\ \Omega$

Discussion of Auxiliary Functions

The auxiliary functions are usually not computable functions. The actors are not robots. And it is not necessary to further define these functions beyond stating their signatures as they are usually performed by human actors. The signature of the inquire license function expresses a possible change to the inquired state. One would think of the inquiring actor somehow noting down, remembering, as it were, which inquiries were attempted or had been made. The signature of the inquire actions function does not express such a state change. But it could be expressed as well.

[17] "Eureka" used to express triumph on a discovery, heuristics

Schema Definitions of Elaboration Functions

10.6 A Transport Contract Language

10.6.1 Narrative

Preparations

In a number of steps (‘A Synopsis’, ‘A Pragmatics and Semantics Analysis’, and ‘Contracted Operations, An Overview’) we arrive at a sound basis from which to formulate the narrative. We shall, however, forego such a detailed narrative. Instead we leave that detailed narrative to the reader. (The detailed narrative can be “derived” from the formalisation.)

A Synopsis

Contracts obligate transport companies to deliver bus traffic according to a timetable. The timetable is part of the contract. A contractor may sub-contract (other) transport companies to deliver bus traffic according to timetables that are sub-parts of their own timetable. Contractors are either public transport authorities or contracted transport companies. Contracted transport companies may cancel a subset of bus rides provided the total amount of cancellations per 24 hours for each bus line does not exceed a contracted upper limit[18]. The cancellation rights are spelled out in the contract[19]. A sub-contractor cannot increase a contracted upper limit for cancellations above what the sub-contractor was told (in its contract) by its contractor[20]. Etcetera.

A Pragmatics and Semantics Analysis

The “works” of the bus transport contracts are two: the timetables and, implicitly, the designated (and obligated) bus traffic. A bus timetable appears to define one or more bus lines, with each bus line giving rise to one or more bus rides. We assume a timetable description along the lines of Appendix G. Nothing is (otherwise) said about regularity of bus rides. It appears that bus ride cancellations must be reported back to the contractor. And we assume that cancellations by a sub-contractor is further reported back also to the sub-contractor’s contractor. Hence eventually that the public transport authority is notified.

Nothing is said, in the contracts, such as we shall model them, about passenger fees for bus rides nor of percentages of profits (i.e., royalties) to be paid back from a sub-contractor to the contractor. So we shall not bother, in this example, about transport costs nor transport subsidies. But will leave that necessary aspect as an exercise.

[18]We do not treat this aspect further in this chapter.
[19]See Footnote 18.
[20]See Footnote 18.

The opposite of cancellations appears to be ‘insertion’ of extra bus rides, that is, bus rides not listed in the time table, but, perhaps, mandated by special events[21] We assume that such insertions must also be reported back to the contractor.

We assume concepts of acceptable and unacceptable bus ride delays. Details of delay acceptability may be given in contracts, but we ignore further descriptions of delay acceptability. but assume that unacceptable bus ride delays are also to be (iteratively) reported back to contractors.

We finally assume that sub-contractors cannot (otherwise) change timetables. (A timetable change can only occur after, or at, the expiration of a license.) Thus we find that contracts have definite period of validity. (Expired contracts may be replaced by new contracts, possibly with new timetables.)

Contracted Operations, An Overview

So these are the operations that are allowed by a contractor according to a contract: (i) *start:* to perform, i.e., to start, a bus ride (obligated); (ii) *cancel:* to cancel a bus ride (allowed, with restrictions); (iii) *insert:* to insert a bus ride; and (iv) *subcontract:* to sub-contract part or all of a contract.

10.6.2 A Formalisation

Syntax

We treat separately, the syntax of contracts (for a schematised example see Page 310) and the syntax of the actions implied by contracts.

Contracts

A concrete example contract can be ‘schematised’:

cid: **contractor** cor **contracts sub-contractor** cee
  to perform operations
    {"start","cancel","insert","subcontract"}
  with respect to timetable tt.

We assume a context (a global state) in which all contract actions (including contracting) takes place and in which the implicit transport net (see Appendix B) is defined.

63. contracts, contractors and sub-contractors have unique identifiers CId, CNm, CNm.

[21]Special events: breakdown (that is, cancellations) of other bus rides, sports event (soccer matches), etc.

64. A contract has a unique identification, names the contractor and the sub-contractor (and we assume the contractor and sub-contractor names to be distinct). A contract also specifies a contract body.
65. A contract body stipulates a timetable and the set of operations that are mandated or allowed by the contractor.
66. An Operation is either a `"start"` (i.e., start a bus ride), a bus ride `"cancel"`lation, a bus ride `"insert"`, or a `"subcontrat"`ing operation.

type
63. CId, CNm
64. Contract = CId × CNm × CNm × Body
65. Body = Op-**set** × TT
66. Op == ″`start`″ | ″`cancel`″ | ″`insert`″ | ″`subcontract`″

An abstract example contract:

(cid,cnm_i,cnm_j,({″`start`″,″`cancel`″,″`insert`″,″`sublicense`″},tt))

Actions

Example actions can be schematised:

(a) cid: **conduct bus ride** (blid,bid) **to start at time** t
(b) cid: **cancel bus ride** (blid,bid) **at time** t
(c) cid: **insert bus ride like** (blid,bid) **at time** t

The schematised license (Page 310) shown earlier is almost like an action; here is the action form:

(d) cid: **sub-contractor** cnm′ **is granted a contract** cid′
to perform operations {"conduct","cancel","insert",sublicense"}
with respect to timetable tt′.

All actions are being performed by a sub-contractor in a context which defines that sub-contractor cnm, the relevant net, say n, the base contract, referred here to by cid (from which this is a sublicense), and a timetable tt of which tt′ is a subset. contract name cnm′ is new and is to be unique. The subcontracting action can (thus) be simply transformed into a contract as shown on Page 310.

type
Action = CNm × CId × (SubCon | SmpAct) × Time
SmpAct = Start | Cancel | Insert
Conduct == mkSta(s_blid:BLId,s_bid:BId)
Cancel == mkCan(s_blid:BLId,s_bid:BId)
Insert = mkIns(s_blid:BLId,s_bid:BId)
SubCon == mkCon(s_cid:CId,s_cnm:CNm,s_body:(s_ops:Op-**set**,s_tt:TT))

examples:

(a) (cnm,cid,mkSta(blid,id),t)
(b) (cnm,cid,mkCan(blid,id),t)
(c) (cnm,cid,mkIns(blid,id),t)
(d) (cnm,cid,
    mkCon(cid′,
        ({″conduct″,″cancel″,″insert″,″sublicense″},tt′),t))

where: cid′ = generate_CId(cid,cnm,t) See Item/Line 69

We observe that the essential information given in the start, cancel and insert action prescriptions is the same; and that the RSL record-constructors (mkSta, mkCan, mkIns) make them distinct.

Contract Identification

67. There is a “root” contract name, rcid.
68. There is a “root” contractor name, rcnm.

value
67 rcid:CId
68 rcnm:CNm

All other contract names are derived from the root name. Any contractor can at most generate one contract name per time unit. Any, but the root, sub-contractor obtains contracts from other sub-contractors, i.e., the contractor. Eventually all sub-contractors, hence contract identifications can be referred back to the root contractor.

69. Such a contract name generator is a function which given a contract identifier, a sub-contractor name and the time at which the new contract identifier is generated, yields the unique new contract identifier.
70. From any but the root contract identifier one can observe the contract identifier, the sub-contractor name and the time that “went into” its creation.

value
69 gen_CId: CId $\times$ CNm $\times$ Time $\rightarrow$ CId
70 obs_CId: CId $\overset{\sim}{\rightarrow}$ CIdL [**pre** obs_CId(cid):cid$\neq$rcid]
70 obs_CNm: CId $\overset{\sim}{\rightarrow}$ CNm [**pre** obs_CNm(cid):cid$\neq$rcid]
70 obs_Time: CId $\overset{\sim}{\rightarrow}$ Time [**pre** obs_Time(cid):cid$\neq$rcid]

71. All contract names are unique.

```
axiom
71 ∀ cid,cid′:CId•cid≠cid′⇒
71   obs_CId(cid)≠obs_CId(cid′) ∨ obs_CNm(cid)≠obs_CNm(cid′)
71   ∨ obs_LicNm(cid)=obs_CId(cid′)∧obs_CNm(cid)=obs_CNm(cid′)
71     ⇒ obs_Time(cid)≠obs_Time(cid′)
```

72. Thus a contract name defines a trace of license name, sub-contractor name and time triple, "all the way back" to "creation".

```
type
   CIdCNmTTrace = TraceTriple*
   TraceTriple == mkTrTr(CId,CNm,s_t:Time)
value
72 contract_trace: CId → LCIdCNmTTrace
72 contract_trace(cid) ≡
72   case cid of
72     rcid → ⟨⟩,
72     _ → contract_trace(obs_LicNm(cid))^⟨obs_TraceTriple(cid)⟩
72   end

72 obs_TraceTriple: CId → TraceTriple
72 obs_TraceTriple(cid) ≡
72   mkTrTr(obs_CId(cid),obs_CNm(cid),obs_Time(cid))
```

The trace is generated in the chronological order: most recent contract name generation times last.

Well, there is a theorem to be proven once we have outlined the full formal model of this contract language: namely that time entries in contract name traces increase with increasing indices.

```
theorem
   ∀ licn:LicNm •
      ∀ trace:LicNmLeeNmTimeTrace • trace ∈ license_trace(licn) ⇒
            ∀ i:Nat • {i,i+1}⊆inds trace ⇒ s_t(trace(i))<s_t(trace(i+1))
```

Semantics

Execution State

Local and Global States Each sub-contractor has an own local state and has access to a global state. All sub-contractors access the same global state. The global state is the bus traffic on the net. There is, in addition, a notion of running-state. It is a meta-state notion. The running state "is made up" from the fact that there are n sub-contractors, each communicating, as contractors, over channels with other sub-contractors. The global state is distinct from sub-contractor to sub-contractor – no sharing of local states between sub-contractors. We now examine, in some detail, what the states consist of.

Global State The net is part of the global state (and of bus traffics). We consider just the bus traffic.

type
133. BusStop == mkBS(s_fhi:HI,s_ol:LI,s_f:Frac,s_thi:HI) 420

183. BusTraffic = T $\overrightarrow{m}$ (N $\times$ (BusNo $\overrightarrow{m}$ (Bus $\times$ BPos))) 425
184. BPos = atHub | onLnk | atBS
185. atHub == mkAtHub(s_fl:LIs_hi:HI,s_tl:LI)
186. onLnk == mkOnLnk(s_fhi:HI,s_ol:LI,s_f:Frac,s_thi:HI)
187. atBSt == mkAtBS(s_fhi:HI,s_ol:LI,s_f:Frac,s_thi:HI)

We shall consider BusTraffic (with its Net) to reflect the global state.

Local sub-contractor contract States: Semantic Types A sub-contractor state contains, as a state component, the zero, one or more contracts that the sub-contractor has received and that the sub-contractor has sublicensed.

type
Body = Op-**set** $\times$ TT
LicΣ = RcvLic$\Sigma\times$SubLic$\Sigma\times$LorBusΣ
RcvLicΣ = LorNm $\overrightarrow{m}$ (LicNm $\overrightarrow{m}$ (Body$\times$TT))
SubLicΣ = LeeNm $\overrightarrow{m}$ (LicNm $\overrightarrow{m}$ Body)
LorBusΣ ... [see below and Page 315] ...

(Recall that LorNm and LeeNm are the same.)

In RecvLics we have that LorNm is the name of the contractor by whom the contract has been granted, LicNm is the name of the contract assigned by the contractor to that license, Body is the body of that license, and TT is that part of the timetable of the Body which has not (yet) been sublicensed.

In DespLics we have that LeeNm is the name of the sub-contractor to whom the contract has been despatched, the first (left-to-right) LicNm is the name of the contract on which that sublicense is based , the second (left-to-right) LicNm is the name of the sublicense, and License is the contract named by the second LicNm.

Local sub-contractor Bus States: Semantic Types The sub-contractor state further contains a bus status state component which records which buses are free, FreeBusΣ, that is, available for dispatch, and where "garaged", which are in active use, ActvBusΣ, and on which bus ride, and a bus history for that bus ride, and histories of all past bus rides, BusHistΣ. A trace of a bus ride is a list of zero, one or more pairs of times and bus stops. A bus history, BusHistory, associates a bus trace to a quadruple of bus line identifiers, bus ride identifiers, contract names and sub-contractor name.[22]

type
  BusNo
  $\mathrm{Bus}\Sigma = \mathrm{FreeBuses}\Sigma \times \mathrm{ActvBuses}\Sigma \times \mathrm{BusHists}\Sigma$
  $\mathrm{FreeBuses}\Sigma = \mathrm{BusStop} \underset{m}{\rightarrow} \mathrm{BusNo}$-**set**
  $\mathrm{ActvBuses}\Sigma = \mathrm{BusNo} \underset{m}{\rightarrow} \mathrm{BusInfo}$
  $\mathrm{BusInfo} = \mathrm{BLId}\times\mathrm{BId}\times\mathrm{LicNm}\times\mathrm{LeeNm}\times\mathrm{BusTrace}$
  $\mathrm{BusHists}\Sigma = \mathrm{Bno} \underset{m}{\rightarrow} \mathrm{BusInfo}^*$
  $\mathrm{BusTrace} = (\mathrm{Time}\times\mathrm{BusStop})^*$
  $\mathrm{LorBus}\Sigma = \mathrm{LeeNm} \underset{m}{\rightarrow} (\mathrm{LicNm} \underset{m}{\rightarrow} ((\mathrm{BLId}\times\mathrm{BId}) \underset{m}{\rightarrow} (\mathrm{BNo}\times\mathrm{BusTrace})))$

A bus is identified by its unique number (i.e., registration) plate (BusNo). We could model a bus by further attributes: its capacity, etc., for for the sake of modelling contracts this is enough. The two components are modified whenever a bus is commissioned into action or returned from duty, that is, twice per bus ride.

Constant State Values There are a number of constant values, of various types, which characterise the "business of contract holders". We define some of these now.

73. For simplicity we assume a constant net — constant, that is, only with respect to the set of identifiers links and hubs. These links and hubs obviously change state over time.
74. We also assume a constant set, leens, of sub-contractors. In reality sub-contractors, that is, transport companies, come and go, are established and go out of business. But assuming constancy does not materially invalidate our model. Its emphasis is on contracts and their implied actions — and these are unchanged wrt. constancy or variability of contract holders.
75. There is an initial bus traffic, tr.
76. There is an initial time, t_0, which is equal to or larger than the start of the bus traffic tr.
77. To maintain the bus traffic "spelled out", in total, by timetable tt one needs a number of buses.
78. The various bus companies (that is, sub-contractors) each have a number of buses. Each bus, independent of ownership, has a unique (car number plate) bus number (BusNo).
    These buses have distinct bus (number [registration] plate) numbers.
79. We leave it to the reader to define a function which ascertain the minimum number of buses needed to implement traffic tr.

[22]In this way one can, from the bus history component ascertain for any bus which for whom (sub-contractor), with respect to which license, it carried out a further bus line and bus ride identified tour and its trace.

value
73. net : N,
74. leens : LeeNm-**set**,
75. tr : BusTraffic, **axiom** wf_Traffic(tr)(net)
76. t_0 : T • $t_0 \geq$ **min dom** tr,
77. min_no_of_buses : **Nat** • necessary_no_of_buses(itt),
78. busnos : BusNo-**set** • **card** busnos $\geq$ min_no_of_buses

79. necessary_no_of_buses: TT $\rightarrow$ **Nat**

80. To "bootstrap" the whole contract system we need a distinguished contractor, named init_leen, whose only license originates with a "ghost" contractor, named root_leen (o, for outside [the system]).
81. The initial, i.e., the distinguished, contract has a name, root_licn.
82. The initial contract can only perform the `"sublicense"` operation.
83. The initial contract has a timetable, tt.
84. The initial contract can thus be made up from the above.

value
80. root_leen,init_ln : LeeNm • root_leen $\notin$ leens $\wedge$ initi_leen $\in$ leens,
81. root_licn : LicNm
82. iops : Op-**set** = {″`sublicense`″},
83. itt : TT,
84. init_lic:License = (root_licn,root_leen,(iops,itt),init_leen)

Initial sub-contractor contract States

type
InitLicΣs = LeeNm $\overrightarrow{m}$ LicΣ
value
ilσ:LicΣ=([init_leen $\mapsto$ [root_leen $\mapsto$ [iln $\mapsto$ init_lic]]]
$\cup$ [leen $\mapsto$ [ ] | leen:LeeNm • leen $\in$ leenms\{init_leen}],[ ],[ ])

Initial sub-contractor Bus States

85. Initially each sub-contractor possesses a number of buses.
86. No two sub-contractors share buses.
87. We assume an initial assignment of buses to bus stops of the free buses state component and for respective contracts.
88. We do not prescribe a "satisfiable and practical" such initial assignment (ibσs).
89. But we can constrain ibσs.
90. The sub-contractor names of initial assignments must match those of initial bus assignments, allbuses.
91. Active bus states must be empty.

92. No two free bus states must share buses.
93. All bus histories are void.

```
type
85. AllBuses′ = LeeNm →m BusNo-set
86. AllBuses = {|ab:AllBuses′•∀ {bs,bs′}⊆rng ab∧bns≠bns′⇒bns ∩ bns′={}|}
87. InitBusΣs = LeeNm →m BusΣ
value
86. allbuses:Allbuses • dom allbuses = leenms ∪ {root_leen} ∧ ∪ rng allbuses = busnos

87. ibσs:InitBusΣs
88. wf_InitBusΣs: InitBusΣs → Bool
89. wf_InitBusΣs(iσs) ≡
90.   dom iσs = leenms ∧
91.   ∀ (_,abσ,_):BusΣ•(_,abσ,_) ∈ rng iσs ⇒ abσ=[ ] ∧
92.   ∀ (fbiσ,abiσ),(fbjσ,abjσ):BusΣ •
92.     {(fbiσ,abiσ),(fbjσ,abjσ)}⊆rng iσs
92.       ⇒ (fbiσ,actiσ)≠(fbjσ,actjσ)
92.         ⇒ rng fbiσ ∩ rng fbjσ = {}
93.           ∧ actiσ=[ ]=actjσ
```

Communication Channels The running state is a meta notion. It reflects the channels over which contracts are issued; messages about committed, cancelled and inserted bus rides are communicated, and fund transfers take place.

Sub-Contractor↔Sub-Contractor Channels Consider each sub-contractor (same as contractor) to be modelled as a behaviour. Each sub-contractor (licensor) behaviour has a unique name, the LeeNm. Each sub-contractor can potentially communicate with every other sub-contractor. We model each such communication potential by a channel. For n sub-contractors there are thus $n \times (n-1)$ channels.

```
channel { l_to_l[ fi,ti ] | fi:LeeNm,ti:LeeNm • {fi,ti}⊆leens ∧ fi≠ti } LLMSG
type LLMSG = ...
```

We explain the declaration: **channel** { l_to_l[fi,ti] | fi:LeeNm, ti:LeeNm • fi≠ti } LLMSG. It prescribes $n \times (n-1)$ channels (where n is the cardinality of the sub-contractor name sets). Each channel is prescribed to be capable of communicating messages of type MSG. The square brackets [...] defines l_to_l (sub-contractor-to-sub-contractor) as an array.

We shall later detail the BusRideNote, CancelNote, InsertNote and FundXfer message types.

Sub-Contractor↔Bus Channels Each sub-contractor has a set of buses. That set may vary. So we allow for any sub-contractor to potentially communicate with any bus. In reality only the buses allocated and scheduled by a sub-contractor can be "reached" by that sub-contractor.

```
channel { l_to_b[l,b] | l:LeeNm,b:BNo • l ∈ leens ∧ b ∈ busnos } LBMSG
type LBMSG = ...
```

Sub-Contractor↔Time Channels Whenever a sub-contractor wishes to perform a contract operation that sub-contractor needs know the time. There is just one, the global time, modelled as one behaviour: time_clock.

```
channel { l_to_t[l] | l:LeeNm • l ∈ leens } LTMSG
type LTMSG = ...
```

Bus↔Traffic Channels Each bus is able, at any (known) time to ascertain where in the traffic it is. We model bus behaviours as processes, one for each bus. And we model global bus traffic as a single, separate behaviour.

```
channel { b_to_tr[b] | b:BusNo • b ∈ busnos } LTrMSG
type
   BTrMSG == reqBusAndPos(s_bno:BNo,s_t:Time) | (Bus×BusPos)
```

Buses↔Time Channel Each bus needs to know what time it is.

```
channel { b_to_t[b] | b:BNo • b ∈ busnos } BTMSG
type
   BTMSG ...
```

Local sub-contractor Bus States: Update Functions

```
value
   update_BusΣ: Bno×(T×BusStop) → ActBusΣ → ActBusΣ
   update_BusΣ(bno,(t,bs))(actσ) ≡
      let (blid,bid,licn,leen,trace) = actσ(bno) in
      actσ†[ bno↦(licn,leen,blid,bid,trace⌢⟨(t,bs)⟩) ] end
      pre bno ∈ dom actσ


value
   update_FreeΣ_ActΣ:
      BNo×BusStop→BusΣ→BusΣ
   update_FreeΣ_ActΣ(bno,bs)(freeσ,actvσ) ≡
      let (_,_,_,_,trace) = actσ(b) in
      let freeσ′ = freeσ†[ bs ↦ (freeσ(bs))∪{b} ] in
      (freeσ′,actσ\{b}) end end
      pre bno ∉ freeσ(bs) ∧ bno ∈ dom actσ
```

```
value
   update_LorBusΣ:
      LorNm×lin:LicNm×len:LeeNm×(BLId×BId)×(BNo×Trace)→LorBusΣ
         → out {l_to_l[len,lorn]|lon:LorNm•lon ∈ leens\{len}} LorΣ
   update_LorBusΣ(lon,lin,len,(bli,bi),(bno,tr))(lbσ) ≡
      l_to_l[len,lon]!Licensor_BusHistΣMsg(bno,bli,bi,lin,len,tr) ;
      lbσ†[len↦(lbσ(len))†[lin↦((lbσ(len))(lin))†[(bli,bi)↦(bno,trace)]]]
      pre len ∈ dom lbσ ∧ lin ∈ dom (lbσ(len))
```

```
value
   update_ActΣ_FreeΣ:
      LeeNm×LicNm×BusStop×(BLId×BId)→BusΣ→BusΣ×BNo
   update_ActΣ_FreeΣ(leen,licn,bs,(blid,bid))(freeσ,actvσ) ≡
      let bno:Bno • bno ∈ freeσ(bs) in
      ((freeσ\{bno},actvσ ∪ [bno↦(blid,bid,licnm,leenm,⟨⟩)]),bno) end
      pre bs ∈ dom freeσ ∧ bno ∈ freeσ(bs) ∧ bno ∉ dom actvσ ∧ ...
```

Run-time Environment So we shall be modelling the transport contract domain as follows: As for behaviours we have this to say. There will be n sub-contractors. One sub-contractor will be initialised to one given license. You may think of this sub-contractor being the transport authority. Each sub-contractor is modelled, in RSL, as a CSP-like process. With each sub-contractor, l_i, there will be a number, b_i, of buses. That number may vary from sub-contractor to sub-contractor. There will be b_i channels of communication between a sub-contractor and that sub-contractor's buses, for each sub-contractor. There is one global process, the traffic. There is one channel of communication between a sub-contractor and the traffic. Thus there are n such channels.

As for operations, including behaviour interactions we assume the following. All operations of all processes are to be thought of as instantaneous, that is, taking nil time ! Most such operations are the result of channel communications either just one-way notifications, or inquiry requests. Both the former (the one-way notifications) and the latter (inquiry requests) must not be indefinitely barred from receipt, otherwise holding up the notifier. The latter (inquiry requests) should lead to rather immediate responses, thus must not lead to dead-locks.

The System Behaviour

The system behaviour starts by establishing a number of licenseholder and bus_ride behaviours and the single time_clock and bus_traffic behaviours

```
value
   system: Unit → Unit
   system() ≡
```

```
licenseholder(init_leen)(il$\sigma$(init_leen),ib$\sigma$(init_leen))
|| (|| {licenseholder(leen)(il$\sigma$(leen),ib$\sigma$(leen))
        | leen:LeeNm•leen $\in$ leens\{init_leen}})
|| (|| {bus_ride(b,leen)(root_lorn,"nil")
        | leen:LeeNm,b:BusNo •leen $\in$ **dom** allbuses $\wedge$ b $\in$ allbuses(leen)})
|| time_clock($t_0$) || bus_traffic(tr)
```

The initial licenseholder behaviour states are individually initialised with basically empty license states and by means of the global state entity bus states. The initial bus behaviours need no initial state other than their bus registration number, a "nil" route prescription, and their allocation to contract holders as noted in their bus states.

Only a designated licenseholder behaviour is initialised to a single, received license.

Semantic Elaboration Functions

The Licenseholder Behaviour

94. The licenseholder behaviour is a sequential, but internally non-deterministic behaviour.
95. It internally non-deterministically ($\lceil\rceil$) alternates between
    (a) performing the licensed operations (on the net and with buses),
    (b) receiving information about the whereabouts of these buses, and informing contractors of its (and its subsub-contractors') handling of the contracts (i.e., the bus traffic), and
    (c) negotiating new, or renewing old contracts.

```
94. licenseholder: LeeNm → (Lic$\Sigma$×Bus$\Sigma$) → **Unit**
95. licenseholder(leen)(lic$\sigma$,bus$\sigma$) ≡
95.     licenseholder(leen)((lic_ops$\lceil\rceil$bus_mon$\lceil\rceil$neg_licenses)(leen)(lic$\sigma$,bus$\sigma$))
```

The Bus Behaviour

96. Buses ply the network following a timed bus route description.
    A timed bus route description is a list of timed bus stop visits.
97. A timed bus stop visit is a pair: a time and a bus stop.
98. Given a bus route and a bus schedule one can construct a timed bus route description.
    (a) The first result element is the first bus stop and origin departure time.
    (b) Intermediate result elements are pairs of respective intermediate schedule elements and intermediate bus route elements.
    (c) The last result element is the last bus stop and final destination arrival time.
99. Bus behaviours start with a "nil" bus route description.

```
type
96. TBR = TBSV*
97. TBSV = Time × BusStop
value
98. conTBR: BusRoute × BusSched → TBR
98. conTBR((dt,til,at),(bs1,bsl,bsn)) ≡
98(a))    ⟨(dt,bs1)⟩
98(b))    ⌢ ⟨(til[i],bsl[i])|i:Nat•i:⟨1..len til⟩⟩
98(c))    ⌢ ⟨(at,bsn)⟩
          pre: len til = len bsl
type
99. BRD == ″nil″ | TBR
```

100. The bus behaviour is here abstracted to only communicate with some contract holder, time and traffic,
101. The bus repeatedly observes the time, t, and its position, po, in the traffic.
102. There are now four case distinctions to be made.
103. If the bus is idle (and a a bus stop) then it waits for a next route, brd′ on which to engage.
104. If the bus is at the destination of its journey then it so informs its owner (i.e., the sub-contractor) and resumes being idle.
105. If the bus is ‘en route’, at a bus stop, then it so informs its owner and continues the journey.
106. In all other cases the bus continues its journey

```
value
100. bus_ride: leen:LeeNm × bno:Bno → (LicNm × BRD) →
100.     in,out l_to_b[leen,bno], in,out b_to_tr[bno], in b_to_t[bno] Unit
100. bus_ride(leen,bno)(licn,brd) ≡
101.   let t = b_to_t[bno]? in
101.   let (bus,pos) = (b_to_tr[bno]!reqBusAndPos(bno,t) ; b_to_tr[bno]?) in
102.   case (brd,pos) of
103.     (″nil″,mkAtBS(_,_,_,_)) →
103.         let (licn,brd′) = (l_to_b[leen,bno]!reqBusRid(pos);l_to_b[leen,bno]?) in
103.         bus_ride(leen,bno)(licn,brd′) end
104.     (⟨(at,pos)⟩,mkAtBS(_,_,_,_)) →
104s         l_to_b[l,b]!BusΣMsg(t,pos);
104          l_to_b[l,b]!BusHistΣMsg(licn,bno);
104          l_to_b[l,b]!FreeΣ_ActΣMsg(licn,bno) ;
104          bus_ride(leen,bno)(ilicn,″nil″),
105.     (⟨(t,pos),(t′,bs′)⟩⌢brd′,mkAtBS(_,_,_,_)) →
105s         l_to_b[l,b]!BusΣMsg(t,pos) ;
105          bus_ride(licn,bno)(⟨(t′,bs′)⟩⌢brd′),
106.     _ → bus_ride(leen,bno)(licn,brd) end end end
```

In formula line 101 of bus_ride we obtained the bus. But we did not use "that" bus ! We we may wish to record, somehow, number of passengers alighting and boarding at bus stops, bus fees paid, one way or another, etc. The bus, which is a time-dependent entity, gives us that information. Thus we can revise formula lines 104s and 105s:

```
Simple:  104s  l_to_b[l,b]!BusΣMsg(pos);
Revised: 104r  l_to_b[l,b]!BusΣMsg(pos,bus_info(bus));

Simple:  105s  l_to_b[l,b]!BusΣMsg(pos);
Revised: 105r  l_to_b[l,b]!BusΣMsg(pos,bus_info(bus));
```

```
type
  Bus_Info = Passengers × Passengers × Cash × ...
value
  bus_info: Bus → Bus_Info
  bus_info(bus) ≡ (obs_alighted(bus),obs_boarded(bus),obs_till(bus),...)
```

It is time to discuss our description (here we choose the bus_ride behaviour) in the light of our claim of modeling "the domain". These are our comments:

- First one should recognise, i.e., be reminded, that the narrative and formal descriptions are always abstractions. That is, they leave out few or many things. We, you and I, shall never be able to describe everything there is to describe about even the simplest entity, operation, event or behaviour.

The Global Time Behaviour

107. The time_clock is a never ending behaviour — started at some time t_0.
108. The time can be inquired at any moment by any of the licenseholder behaviours and by any of the bus behaviours.
109. At any moment the time_clock behaviour may not be inquired.
110. After a skip of the clock or an inquiry the time_clock behaviour continues, non-deterministically either maintaining the time or advancing the clock!

```
value
107. time_clock: T →
107.       in,out {l_to_t[leen] | leen:LeeNm • leen ∈ leenms}
107.       in,out {b_to_t[bno] | bno:BusNo • bno ∈ busnos}  Unit
107.  time_clock:(t) ≡
109.    (skip ⌈⌉
108.    (⌈⌉{l_to_t[leen]? ; l_to_t[leen]!t | leen:LeeNm•leen ∈ leens})
108.    ⌈⌉ (⌈⌉{b_to_t[bno]? ; b_to_t[bno]!t | bno:BusNo•bno ∈ busnos})) ;
110.    (time_clock:(t) ⌈⌉ time_clock(t+$\delta_t$))
```

The Bus Traffic Behaviour

111. There is a single bus_traffic behaviour. It is, “mysteriously”, given a constant argument, “the” traffic, tr.
112. At any moment it is ready to inform of the position, bps(b), of a bus, b, assumed to be in the traffic at time t.
113. The request for a bus position comes from some bus.
114. The bus positions are part of the traffic at time t.
115. The bus_traffic behaviour, after informing of a bus position reverts to “itself”.

```
value
111. bus_traffic: TR → in,out {b_to_tr[bno]|bno:BusNo•bno ∈ busnos} Unit
111. bus_traffic(tr) ≡
113.   ⌷ { let reqBusAndPos(bno,time) = b_to_tr[b]? in assert b=bno
112.         if time ∉ dom tr then chaos else
114.         let (_,bps) = tr(t) in
112.         if bno ∉ dom tr(t) then chaos else
112.         b_to_tr[bno]!bps(bno) end end end end | b:BusNo•b ∈ busnos} ;
115.   bus_traffic(tr)
```

License Operations

116. The lic_ops function models the contract holder choosing between and performing licensed operations.
     We remind the reader of the four actions that licensed operations may give rise to; cf. the abstract syntax of actions, Page 311.
117. To perform any licensed operation the sub-contractor needs to know the time and
118. must choose amongst the four kinds of operations that are licensed. The choice function, which we do not define, makes a basically non-deterministic choice among licensed alternatives. The choice yields the contract number of a received contract and, based on its set of licensed operations, it yields either a simple action or a sub-contracting action.
119. Thus there is a case distinction amongst four alternatives.
120. This case distinction is expressed in the four lines identified by: 120.
121. All the auxiliary functions, besides the action arguments, require the same state arguments.

```
value
116. lic_ops: LeeNm → (LicΣ×BusΣ) → (LicΣ×BusΣ)
116. lic_ops(leen)(licσ,busσ) ≡
117.   let t = (time_channel(leen)!req_Time;time_channel(leen)?) in
118.   let (licn,act) = choice(licσ)(busσ)(t) in
119.   (case act of
120.      mkCon(blid,bid)  → cndct(licn,leenm,t,act),
120.     mkCan(blid,bid)   → cancl(licn,leenm,t,act),
120.     mkIns(blid,bid)   → insrt(licn,leenm,t,act),
120.     mkLic(leenm′,bo) → sublic(licn,leenm,t,act) end)(licσ,busσ) end end

        cndct,cancl,insert: SmpAct→(LicΣ×BusΣ)→(LicΣ×BusΣ)
        sublic: SubLic→(LicΣ×BusΣ)→(LicΣ×BusΣ)
```

Bus Monitoring Like for the bus_ride behaviour we decompose the bus_monitoring behaviour into two behaviours. The local_bus_monitoring behaviour monitors the buses that are commissioned by the sub-contractor. The licensor_bus_monitoring behaviour monitors the buses that are commissioned by sub-contractors sub-contractd by the contractor.

```
value
   bus_mon: l:LeeNm → (LicΣ×BusΣ)
            → in {l_to_b[l,b]|b:BNo•b ∈ allbuses(l)} (LicΣ×BusΣ)
   bus_mon(l)(licσ,busσ) ≡
      local_bus_mon(l)(licσ,busσ) ⌈⌉ licensor_bus_mon(l)(licσ,busσ)
```

122. The local_bus_monitoring function models all the interaction between a contract holder and its despatched buses.
123. We show only the communications from buses to contract holders.
124. Etcetera.

```
122. local_bus_mon: leen:LeeNm → (LicΣ×BusΣ)
123.       → in {l_to_b[leen,b]|b:BNo•b ∈ allbuses(l)} (LicΣ×BusΣ)
122. local_bus_mon(leen)(licσ:(rlσ,slσ,lbσ),busσ:(fbσ,abσ)) ≡
124.    let (bno,msg) = ⌈⌉{(b,l_to_b[l,b]?)|b:BNo•b ∈ allbuses(leen)} in
124.    let (blid,bid,licn,lorn,trace) = abσ(bno) in
124.    case msg of
124.      BusΣMsg(t,bs) →
124.        let abσ′ = update_BusΣ(bno)(licn,leen,blid,bid)(t,bs)(abσ) in
124.        (licσ,(fbσ,abσ′,histσ)) end,
124.      BusHistΣMsg(licn,bno) →
124.        let lbσ′ = update_LorBusΣ
124.           (obs_LorNm(licn),licn,leen,(blid,bid),(b,trace))(lbσ)  in
124.        l_to_l[leen,obs_LorNm(licn)] !
124.           Licensor_BusHistΣMsg(licn,leen,bno,blid,bid,tr);
124.        ((rlσ,slσ,lbσ′),busσ) end
124.      FreeΣ_ActΣMsg(licn,bno) →
124.        let (fbσ′,abσ′) = update_FreeΣ_ActΣ(bno,bs)(fbσ,abσ) in
124.        (licσ,(fbσ′,abσ′)) end
124.    end end end
```

125. Reader is to provide the narrative!

```
125. licensor_bus_mon: lorn:LorNm → (LicΣ×BusΣ)
125.   → in {l_to_l[lorn,leen]|leen:LeeNm•leen ∈ leenms\{lorn}}
125.     (LicΣ×BusΣ)
125. licensor_bus_mon(lorn)(licσ,busσ) ≡
125.     let (rlσ,slσ,lbhσ) = licσ in
125.     let (leen,Licensor_BusHistΣMsg(licn,leen″,bno,blid,bid,tr))
125.       = ⌈⌉{(leen′,l_to_l[lorn,leen′]?)|leen′:LeeNm•leen′ ∈ leenms\{lorn}} in
125.     let lbhσ′ =
125.       update_BusHistΣ
125.       (obs_LorNm(licn),licn,leen″,(blid,bid),(bno,trace))(lbhσ) in
125.     l_to_l[leenm,obs_LorNm(licnm)] !
125.       Licensor_BusHistΣMsg(b,blid,bid,lin,lee,tr);
125.     ((rlσ,slσ,lbhσ′),busσ)
125.     end end end
```

The Conduct Bus Ride Action

126. The conduct bus ride action prescribed by (ln,mkCon(bli,bi,t′) takes place in a context and shall have the following effect:
   (a) The action is performed by contractor li and at time t. This is known from the context.
   (b) First it is checked that the timetable in the contract named ln does indeed provide a journey, j, indexed by bli and (then) bi, and that that journey starts (approximately) at time t′ which is the same as or later than t.
   (c) Being so the action results in the contractor, whose name is "embedded" in ln, receiving notification of the bus ride commitment.
   (d) Then a bus, selected from a pool of available buses at the bust stop of origin of journey j, is given j as its journey script, whereupon that bus, as a behaviour separate from that of sub-contractor li, commences its ride.
   (e) The bus is to report back to sub-contractor li the times at which it stops at en route bus stops as well as the number (and kind) of passengers alighting and boarding the bus at these stops.
   (f) Finally the bus reaches its destination, as prescribed in j, and this is reported back to sub-contractor li.
   (g) Finally sub-contractor li, upon receiving this 'end-of-journey' notification, records the bus as no longer in actions but available at the destination bus stop.

The Cancel Bus Ride Action

127. The cancel bus ride action prescribed by (ln,mkCan(bli,bi,t′) takes place in a context and shall have the following effect:

(a) The action is performed by contractor li and at time t. This is known from the context.
(b) First a check like that prescribed in Item 126(b)) is performed.
(c) If the check is OK, then the action results in the contractor, whose name is "embedded" in ln, receiving notification of the bus ride cancellation.
That's all !

The Insert Bus Ride Action

128. The insert bus ride action prescribed by (ln,mkIns(bli,bi,t′) takes place in a context and shall have the following effect:
    (a) The action is performed by contractor li and at time t. This is known from the context.
    (b) First a check like that prescribed in Item 126(b)) is performed.
    (c) If the check is OK, then the action results in the contractor, whose name is "embedded" in ln, receiving notification of the new bus ride commitment.
    (d) The rest of the effect is like that prescribed in Items 126(d))–126(g)).

The Contracting Action

129. The subcontracting action prescribed by (ln,mkLic(li′,(pe′,ops′,tt′))) takes place in a context and shall have the following effect:
    (a) The action is performed by contractor li and at time t. This is known from the context.
    (b) First it is checked that timetable tt is a subset of the timetable contained in, and that the operations ops are a subset of those granted by, the contract named ln.
    (c) Being so the action gives rise to a contract of the form (ln′,li,(pe′,ops′,-tt′),li′). ln′ is a unique new contract name computed on the basis of ln, li, and t. li′ is a sub-contractor name chosen by contractor li. tt′ is a timetable chosen by contractor li. ops′ is a set of operations likewise chosen by contractor li.
    (d) This contract is communicated by contractor li to sub-contractor li′.
    (e) The receipt of that contract is recorded in the license state.
    (f) The fact that the contractor has sublicensed part (or all) of its obligation to conduct bus rides is recorded in the modified component of its received contracts.

10.6.3 Discussion

10.7 Conclusion

It really is too early — in the development of the topic of this chapter — to conclude!

10.7.1 Achievements

What Did We Wish to Achieve?

Or rather, at this early, incomplete stage, what do we wish to achieve? In a first round we wish to achieve the following: an understanding of different kinds of license languages; an understanding of obligations and permissions (yet to be "designed" more explicitly into the three languages; a formalisation of both common aspects of the license systems (as a "vastly" distributed set of very many actors acting on even more licenses "competing" for resources, etc.), as well as of each individual language.

What Have We Achieved?

We think we have achieved what we set out to achieve.

What Do We Now Wish to Achieve?

First we would like to complete the full formalisation of each of the four languages: three license languages and one contract language. Based on those four formalisations we hope to be able to identify some common, better formalised, i.e., parametrised, license and contract concepts and thus to "lift" the four sets of syntaxes, well-formedness predicates and semantic functions into one set of parametrised functions and syntaxes. We think that given such four, widely separate examples and their parametrised "lifting" we can offer better contract and license language design, parametrised formalisations and common parametrised implementation software designs.