RailCons CNL at Chalmers

Bjørnar Luteberget

2016-11-02

Formalizing "Teknisk regelverk"

The Norwegian railway authority has a comprehensive set of technical regulations ("Teknisk regelverk") for design/planning ("prosjektering"), construction, and maintenance of railway infrastructure. The text is also subdivided into 12 disciplines, such as electric, track works, tunnels, and signalling.

Goal

The RailCons project would like to formalize relevant parts of the technical regulations for use in the on-the-fly type of verification which can be used within railway construction design software (specifically the RailCOMPLETE software). This type of verification will probably be limited to static infrastructure analysis, leaving the more heavy-weight analysis of e.g. the implementation of control systems to specialized analysis software such as Prover AB (Sweden) or Systerel (France). However, it could still be beneficial for the method for formalizing the technical regulations to include more dynamic and logically more complex kinds of information, as this could be used to analyse the regulations in themselves, or as input to other types of analysis software.

The suggested use cases are listed here by priority:

1. On-the-fly verification inside the design tool for **static infrastructure**, probably based on the Datalog logic. Related sub-goals are:
    1. Engineers can understand what the verification engine is doing. (CNL + CAD)
    2. Produce reports on how/what is verified. (CNL + CAD with report generator)
    3. Regulations are changing, so regulations database needs to be maintained. (CNL editor)
    4. Expert knowledge, rules of thumb, local company knowledge base. (CNL editor)
2. Control system **implementation verification**, typically more computationally expensive, and based on other formalisms, such as timed automata, first order logic, etc.
3. Verification of human **activities and processes**, such as regulations for maintenance scheduling, or regulations for design *considerations*. (For example, a non-standard design choice for infrastructure could be acceptable only when reasoning has been given for abandoning the standard choice.)
4. Analysis of the regulations in themselves, for example exposing inconsistencies.
5. Tranforming the regulations into a simpler, unambiguous text for human readers.

Regulations overview

The technical regulations (*"Teknisk regelverk"*) can be found at https://trv.jbv.no/ and consists of the following books:

- **Common regulations**: 501 Common regulations
- **Common electrical**: 510 Design and construction
- **Signs**: 515 Placement of signs along the track
- **Superstructure (tracks)**: 530 Design, 531 Construction, 532 Maintenance
- **Substructure**: 520 Design and construction, 522 Maintenance
- **Tunnels**: 521 Design and construction, 523 Maintenance
- **Bridges**: 525 Design and construction, 527 Maintenance
- **Overhead line**: 540 Design, 541 Construction, 542 Maintenance
- **Low voltage and 22 kV**: 543 Design, 544 Construction, 545 Maintenance
- **Power supply**: 546 Design, 547 Construction, 548 Maintenance
- **Signalling**: 550 Design, 551 Construction, 552 Maintenance, 553 Assessment
- **Telecommunications**: 560 Design and construction, 562 Maintenance

Structure of each book:

- Each book repeats the *common regulations* as the first three chapters.
- Following this will typically be a *general* section containing:
  - declaration of the scope of the book,
  - references to relevant standards,
  - definitions of relevant techincal terms,
  - qualitative classifications, such as quality classes, risk classes, etc.
- The main part of a book consists typically of 5 to 10 chapters, each detailing a specific technical topic within the discipline. The text consists of:
  - Scope declarations
  - Definitions
  - Non-normative statements
  - Comments
  - Regulations (including tables and figures), with exceptions
  - Examples

Scope and focus

The technical regulations contain a lot of generalities which are not necessarily normative, nor directly useful in a design setting. Based on the prioritized use case list, the following parts of the regulations should be considered first in designing and testing the formalization procedure:

1. Superstructure design (track design / *Overbygning: 530 Prosjektering*), especially regulations and formulas regarding
   - track geometry: curvature, gradients, etc.
   - switches: types, maximum speeds, naming (numbering), etc.
2. Signalling design (*Signal: 550 Prosjektering*), especially
   - signal placement, functions, sighting distance
   - train detector placement, classification
   - switch motors requirements and control system components
   - automatic train protection system (ATC) placement and functions

- interlocking (control system) routes, conflicts, detection sections, safety classes, flank protection, overlaps, etc.

Rule classification

This is an attempt to classify rule types, and could help determine what the language needs to support, and how the railway language is distinguished from other CNLs and general ontology languages.

- **Classes and properties**, descriptions of objects with (complex) classification and properties. Properties are typically
  - Picked from enumeration (enumerations defined by an ontology)
  - Integers or floating point numbers
  - String operations (not so important so far, not directly supported in Datalog).

  See the RAINS CNL or any other ontology-based CNL.
- **Formulas** mathematical type, usually also supported by ontology-like languages.
- **Topology** specifies graph searches using the (somewhat expensive) built-in predicates such as *connected*, *following*, *between*, *overlap*.
- **Geometry** constraint language (as suggested by Gerardo).
- *Future:* **LTL/CTL** style model checking properties. Any examples of CNLs for this?
- *Future:*

Tables are not a class of rules, only syntactic sugar for facts.

Translation

The technique of controlled natural language will be investigated to allow the input of regulations, especially those translatable into the static infrastructure verification being developed in the RailCons project.

Non-textual information

It could be beneficial to keep tables and figures from the regulations when transitioning to the CNL, so that the CNL interfaces can use these kinds of rich text.

Specifically, referring to figures and referring to the contents of tables based on column and row headers, could be useful.

Examples and relevance

The following table lists some example excerpts from the regulations along with a translation into English, and a comment about use cases and relevance.

Use cases	Original text	English translation	Comments
Overbygning: 530 Prosjektering, Kap. 8 Helsveist spor, 2.1.			
#4	De store krefter som kan forekomme i et helsveist spor stiller strenge krav til sporets konstruksjon.	The large forces that may occur in a welded track makes stringent demands on the track construction.	This sentence is not normative, and is unlikely to have any use in automated verification.
Overbygning: 530 Prosjektering, Kap. 8 Helsveist spor, 2.1.3 a)			
#4	Ballasten skal på linjen og i hovedspor på statsjoner være fullverdig grovpukk (av størrelse 31.5 – 63 mm)	The ballast on the line and in the main track at stations must be purely coarse crushed stone (size from 31.5 to 63 mm)	This is a specification which is absolute, and rules out the need for specifying this as a part of the design, because it is not part of a specific station. It can still be valuable to support this sentence in a CNL, and in a formal representation.
Overbygning: 530 Prosjektering, Kap. 8 Helsveist spor, 2.1.2 a)			
#1	Minste kurveradius for helsveist med betongsviller skal være 250 m.	The lowest allowable radius of curvature for whole welded track on concrete sleepers is 250 m.	This is a typical example of static infrastructure verification, expressible in Datalog as: error(Segment) :- trackSegment(Segment), trackSegmentRadius(Segment, Radius), Radius < 250.
Signal: 550 Prosjektering, Kap. 6 Lyssignal, 2.1.2 j)			
#1	Et innkjørhovedsignal skal plasseres $\geq$ 200 meter foran innkjørtogveiens første sentralstilte, motrettede sporveksel, se Figur 5.	A home main signal shall be placed at least 200 m in front of the first controlled, facing switch in the entry train path (see Figure 5).	This is the example that we have been using most frequently for the RailCons verification tool. Datalog: error(Sig,Sw) :- firstFacing(Bdry, Sw, Dir), homeSignalBetween(Sig, Bdry, Sw), distance(Sig, Sw, Dir, L), L < 200.
Signal: 550 Signal, Kap. 5 Forriglingsutrustning, 2.8.1 Dekningsgivende objekt			

Use cases	Original text	English translation	Comments
#1, #2	Følgende objekt kan være dekningsgivende: Hovedsignal, Dvergsignal, Sporveksel, Sporsperre, Avsporingstunge, Signal E35 Stoppskilt. Et hovedsignal skal vise signal ”Stopp” for å være dekningsgivende.	The following objects can provide flank protection: main signal, shunting signal, switch, derailer, derailing tongue, signal E35 stop sign. A main signal must display "stop" to provide flank protection.	This regulation is relevant both for **specifying** the control system, and for verifying the **implementation**. The specification chooses which objects to use for flank protection (**static**) and what state they can be used in, while the implementation must correctly enforce the conditions saying which message the signal displays (**dynamic**).
Signal: 550 Prosjektering, Kap. 6 Lyssignal, 2.1.2 i)			
#1, #3	Et hovedsignal bør ikke plasseres i tunneler, på bruer, eller andre steder hvor en eventuell togstans og dermed muligheten for avstigning, vil medføre fare.	A main signal should not be placed in tunnels, on bridges, or other places where halting trains and thus the possibility of disembarking, can impose dangers.	Here we have an example of a “*should*” modality, where the static infrastructure verification could issue a warning, but not an error. Also, it could be required to document the alternatives that were considered when deciding on the design.
Signal: 550 Prosjektering, Kap. 5 Forriglingsutrustning, 4.1.1.1 i)			
#2	For at en togvei skal kunne fastlegges, skal et objekt som gir dekning til togveien være dekningsgivende.	For a train route to be deactivated, any object giving flank protection must be in a protecting state.	This regulation concerns only the state of the control system, and as such relates to the implementation of the control system and not the static infrastructure specification.
Overbygning: 530 Prosjektering, Kap. 5 Sporets trasé, 3.1 Dimensjonerende parametre			

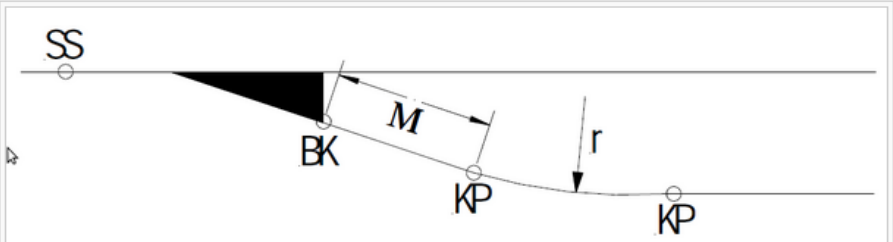
Use cases	Original text	English translation	Comments
#1	See table below.	(a) minimum radius, (b) maximal superelevation, (c) limit on superelevation cause by derailment risk at low speeds, (d) limit for superelevation rate of change, (e) limit for superelevation deficit.	Limiting values are organized in a table for use in formulas in other sections.

Tabell 2: Dimensjonerende parametre for nye baner og linjeomlegginger

	Symbol	Definisjon	Normale krav	Minste krav
a)	$R_{\min}$	minste radius	250 m	190 m
b)	h_{maks}	maksimal verdi for overhøyden [1)]	150 mm	
c)	h_{avsp}	grense for overhøyde pga. avsporingsfaren ved lave hastigheter	$\frac{R-100}{2}$ mm	
d)	$\left(\frac{\Delta h}{L}\right)_{maks}$	grenseverdi for rampestigning	1:400	
e)	I_{maks}	grenseverdi for manglende overhøyde [2)]	R ≤ 600: 100 mm R > 600: 130 mm	

Overbygning: 530 Prosjektering, Kap. 5 Sporets trasé, 3.7 Sporveksler og sporforbindelser

#1	Avstanden mellom sporveksel og overgangskurve, sirkelkurve, bru eller annen motstående sporveksel skal ikke være mindre enn avstanden M gitt i *Kurver uten overgangskurver*, krav b). M skal imidlertid ikke være kortere enn 6 m.	The distance between the switch point and the transition curve, circle curve, bridge or other opposite switch point should not be less than the distance M given in section *"curves without transition curves"*, requirements b). M shall not be shorter than 6 m.	The parameter M is explained by the figure below. Reference is given to another section of the regulations.



Figur 4: Krav til rettlinje M ved linjedelende sporveksel

Overbygning: 530 Prosjektering, Kap. 5 Sporets trasé, 5 Største hastighet – sporets geometri

Use cases	Original text	English translation	Comments
#1	Hastigheten i en kurve skal ikke være større enn: $V = 0,291 \cdot \sqrt{R\,(h + I_{\text{maks}})}$ (5) Hvis ligning 5 i tilfeller med falsk overhøyde gir lavere verdi enn 20 km/h gjelder $V = 20$ km/h.	The speed in a curve shall not exceed: $V = 0,291 \cdot \sqrt{R\,(h + I_{\text{maks}})}$ (5) If Eq. 5 gives a lower value than 20 km/h in situations with false superelevation, then $V = 20$ km/h shall be used.	Use of equations with designed and given parameters.
Signal: 552 Vedlikehold, Kap. 6 Lyssignal, 3 Lyssignaler			
#3, #4	Dersom lyssignal er vridd eller på annen måte kommet ut av stilling skal dette utbedres snarest.	If a signal is twisted or in other ways are out of position, this shall be fixed as soon as possible.	Typical maintenance regulation. Here, it might be sufficient to identify this as a *checklist item*, for maintenance scheduling and reporting purposes.

Translation toolchain

Suggest the following steps in transferring *teknisk regelverk* into Datalog.

1. **Original**: *Teknisk regelverk* in its original form
2. **Categorization**: Non-automatic or assisted transfer from original text to a categorization of each part (sentence?) of the text into the following:
   - Definitions
   - Scopes
   - Rules
   - Figures
   - Tables
3. **High level CNL**: Non-automatic or assisted transfer of the definitions and rules in the categorized text into a high-level CNL, with Norwegian-like and/or English-like concrete syntax, supporting the kind of modalities and logic in the target (Datalog) language, with special constructs for the railway domain.
4. **Low-level CNL**: Fully automatic transfer from the high-level CNL into a format which is specially suited for transfer to Datalog. Rules can be in the following form:

   ```
   Given "FACT_1" and "FACT_2" ..., then
   "COND_1" and "COND_2" ..., should be satisfied.
   ```

5. **Datalog**: final output format of the translation

High level CNL

The high level CNL should:

- have Norwegian-like and/or English-like concrete syntax,
- recognize several types of top-level sentence types:
  - definition, obligation, constraint, suggestion, heuristic, etc.
- allow logic which is easily translatable to the target (Datalog) language
  - conjunction, negation, arithmetic, implication(?)
- support special constructs for the railway domain.

Top-level sentence types

Several kinds of sentences:

- **Definition** defines something which can be referred to by other sentences. Example:

  ```
  The minimum radius R_min for use in formulas is 200 m.
  ```

- **Obligation** is a
- **Constraint** is an absolute demand, possibly on the static infrastructure. Example:

  ```
  The minimum radius for any track on a new station is R_min.
  ```

- **Suggestion** is a *"should"* statement.

  ```
  A main signal should not be placed in tunnels.
  ```

- **Heuristic** might be used to give a suggestion for missing information, but should not override any specified information.
- **Comments** are maybe not worth representing?

Low-level logic

The top-level sentence types define which parts the sentence is made up of, while the low-level logic can be

- **Conjunction**, example:

  ```
  A signal shall belong to a track and have a direction.
  ```

- **Negation**, example:

  ```
  A signal shall not be taller than 9 m.
  ```

- **Arithmetic**:
- **Implication**.

Railway constructs

To give more natural-ness to sentences, a vocabulary of railway terms will be "baked in" to the language. Some suggestions:

- Any object type known in the object model for static infrastructure should be a recognized *noun* in the natural language, and the presence an instance of this noun (`a signal`) is translated into an "individual" with the corresponding classification. I.e., `a signal` will be translated into the unary predicate `signal(_0)` in the final representation.
- Known relations should be complemented with specific natural language, such as:
  - `distance(a,b,l)`

    —> `the distance from a to b is l`
  - `distance(a,b,l), l < 50`

    —> `the distance from a to b is less than 50 m`
  - `track(X)`

    —> `X is a track` (for any known object type)
  - `track(X), trackQualityClass(X,A)`

    —> `X is a track of quality class A`
  - `boundary(B), firstSameDirSwitch(B,Sw), facingSwitch(B,Sw), controlled(Sw)`

    —> `The first facing, controlled switch after a station boundary`

Other natural-ness considerations

- Reflection from facts to conditions, for example:
  - `If X is a track with quality class A, then X must have a radius higher than 500 m in all segments`

  is written as
  - `Tracks of quality class A must have radius higher than 500 m in all segments`
- From the GF book, chapter 6, semantic actions:
  - *Anaphroric* expressions:

    `If a man walks he talks`
  - *Aggregation* transfer:

    `John runs or John walks --> John runs or walks`
- From the GF book, chapter 8, interfacing formal and natural languages:
  - Natural language generation to/from logic can in general require steps that are outside of the grammar. However, some possibilities exist for improving naturalness using GF directly.

The fact that Datalog disallows nested predicates (function symbols), simplifies some aspects of linearization into natural language. For example, the negation of arbitrary sentences (as presented on p. 190 in the GF book) is not necessary, as negation is only possible in body literals, which cannot be nested. More specifically, since literals can have the clause type `Cl` instead of the sentence type `S`, natural expressions of negation are available directly from the resource grammar.

Whenever negation required over a complex term is needed, it is required to define an intermediate Datalog predicate whose atomic representation can be negated instead of the complex. This avoid the (usually less elegant) sentence construction "it is not the case that …".

These considerations have led to the comparison section below.

Modules

This is the current structure of the RailCNL GF grammar.

- RailCNL (main)
- Datalog
- Ontology
  - Classes
  - Properties (=, != <, >)
  - Combine these into selections (subject, object, condition)
  - Constr./obl./rec. that selections have properties.
- Graph
  - All/exists path from SEL to (first) SEL must/should pass SEL (SEL is a selection from the Ontology grammar)
  - Distance from SEL to SEL must be RESTR

Other modules/features

Further grammar support could include the following.

- Object placement
  - On bridge, in tunnel, etc.
  - "Where freight trains brake"
- Sighting
- Static driving constraints
  - Must not pass more than 4 balise groups per second (Define static velocities (sign, atc, max static, max dynamic))
  - Safety zones, residual energy, potential consequences
- Track design
- Constructions (geometry)
  - Platforms (relate signalling to constructions)
  - "Things in the way".
- Catenary design (+ return power, grounding/earthing)
- Availability, robustness in components

Strategies for translating HL to LL

The **low level** (LL) language has a grammar which is close to Datalog, adding only the distinction between rules (definition, constraint, etc.), and metadata about rules (name, ID, extended description, severity, etc.), which was organized as *structured comments* in the prototype (from iFM). It linearizes either directly to Datalog, or to a (somewhat less natural) CNL.

The **high level** (HL) language adds constructs as described in the sections above. The translation from HL language to LL language should be deterministic.

A few strategies have come to mind:

Basic GF resource grammar use

- CNL : "`if X is a macroscopic_node, then X is a station_boundary`".
- Write resource grammar linearization (natural language AST is only implicit).
- AST : "`ifthen(macroscopic_node(X),station_boundary(X)).`"

It is not directly possible to make variables implicit in this approach. (From chapter 8 of the GF book: *Natural language generation to/from logic can in general require steps that are outside of the grammar.*)

Montague grammars in GF

Approach described in *Aarne Ranta, "Computational Semantics in Type Theory"*.

- CNL : "`all women walk`".
- Use resource grammar to convert sentence into natural language AST (explicit resource grammar AST).

  `Cl(Pred VP (Det CN woman_N [....]))`
- Use a transfer function to *interpret* the sentence into a logic. `All (\x -> If (iN woman_N x) (iV walk_x)).`

Requires higher-order types in GF, and writing a transfer function. Not necessarily a two-way approach.

AST transfer functions

Generalizing the above.

- HL CNL : "`A macroscopic node is [also] a station boundary.`".
- Write an AST and resource grammar linearization which supports all of the interesting constructs. Here:

  `SubClass(NounModifier(macroscopic, node), NounModifier(station, boundary)).`
- Write transfer functions from HL to LL, optionally back again.

  `macroscopic_node(X) -> station_boundary(X).`
- Linearize to LL.

AST rewriting system

Converting from LL to HL might be complicated. Straight-forwardly writing an explicit transfer function from LL to HL might not be feasible. Also, if we consider the HL grammar an (extensible) extension of the LL grammar, then there might be several *stages of refinement* going from the LL language to the HL language. These factors suggest that a term rewriting system (rule-based) might be suitable. In this case, we should be able to show that the LL language is a normal form.

It might not be a top priority to convert from LL to HL, but it could also be used to illustrate how extending the HL with better constructs improves the naturalness of the language.

See *Ranta: Translating between language and logic* for an overview of rewriting rules used in a first order logic natural language generation system, going from the low level logic representation to a higher level AST which generates more natural sentences.

Sentence translation case studies

Object type constraints

Let's start with the following equivalent sentences:

- Datalog:

```
error_trackquality(X) :- track(X), !trackQualityClass(X,k0).
```

- Low level, in Datalog style

```
<constraint id="trackquality" params="X">
   track(X), !trackQualityClass(X,k0)
</constraint>
```

- Low level, in CNL style

```
<constraint id="trackquality" params="X">
   Given {X is a track}, then
   {quality class of X is k0} should be satisifed.
</constraint>
```

- High level CNL could be any of these:
  - `For any track X, the quality class of X must be k0`
  - `All tracks must have quality class k0.`
  - `Every track must have quality class k0.`
  - `A track shall have quality class k0.`
  - `A track must have quality class k0.`

A structure (AST) for this constraint could be

```
Constraint = Constraint [Clause] [Clause].
Clause = Clause Predicate [Parameter].
Predicate = Predicate String.
Parameter = Constant String | Variable String.
```

The actual abstract representation we want to express is then the following:

```
Constraint [Clause (Predicate "track") [Variable "X"]]
           [Clause (Predicate "trackQualityClass") [Variable "X", Constant "k0"]]
```

Instead of the very general "*predicate*", we could instead use classification, property, association, etc. primitives, so that we get something like:

```
Constraint [Class "track" (Variable "X")]
           [Property "qualityClass" (Variable "X") (Constant "k0")]
```

This could maybe help to restrict predicates to one and two parameters for user-defined concepts, while allowing special internally defined predicates, like *distance*, *between*, etc. to have natural linearizations. This could be achieved through specialization of the grammar for these concepts, instead of supporting arbitrary higher-arity relations.

Other examples

Classification

```
A *macroscopic node* is considered a *station boundary*.
```

Definition (search)

```
A *default train route* consists of two main signals Sa and Sb,
both oriented in the direction Dir, such that Sb follows Sa and
there is a path without signals from Sa to Sb in direction Dir.
```

Object properties

```
All signal should have a type. / Alle signaler skal være av en signaltype.
```

Must placement

The placement of the word must in the sentence determines where the subject ends and the condition starts. Moving it also affects the number of rules that must be generated.

Examples:

- *If a track has closest balise, the balise must be red.*

  ```
  track(X), balise(A), closest_balise(X,A), !red(A) -> error
  ```

- *All tracks must have a closest balise which is red.*

  ```
  track(X), balise(A), !closest_balise(X,A)          -> error

  track(X), balise(A),  closest_balise(X,A), !red(A) -> error
  ```

- *A track which has a closest balise which is red, must exist.*

```
!track(X) -> error

track(X), balise(A), !closest_balise(X,A) -> error

track(X), balise(A), closest_balise(X,A), !red(A) -> error
```

All of these examples also requires the (graph-domain) definition of `closest_balise`.

And/or and negation

The condition part of a sentence is negated in the rule. As Datalog does not (necessarily) have an *or* operator, nor negation over complex terms, these must be given special consideration.

- *Et hovedsignal bør ikke plasseres i tunneler eller på broer.*

```
Heuristic (Property (Class Signal) (Adjective Main))
  (Not (Or (Placement Tunnel) (Placement Bridge)))

signal(X), type(X,main), placement(X,tunnel) -> warning
signal(X), type(X,main), placement(X,bridge) -> warning
```

- *Et hovedsignal bør festes i mast eller åk.*

```
Heuristic (Property (Class Signal) (Adjective Main))
  (Or (Mounting Mast) (Mounting Yoke))

signal(X), type(X,main), !should_mounting_rule123(X) -> warning
mounting(X, mast) -> should_mounting_rule123(X).
mounting(X, yoke) -> should_mounting_rule123(X).
```

System overview

A verification and knowledge base editor system integrated

- Basic low-level grammar (GF, static)
- Ontology / vocabulary based on railML or RailCOMPLETE config, from C# classes, XSD or OWL ontology or similar. (could be static)
  - Could be extended with CNL-related information such as noun genders.
- High-level grammar extensions (GF, static)
  - Ontology (classes and properties)
  - Graph searches (“first”, “all paths”, “some paths”, etc.)
- Dynamically defined concepts.
- Dynamic evaluation/testing environment.

Dynamic

In the KeY book chapter by Johannisson, it seems that he has used a conversion from UML to GF, then the GF compiler, and combined this with the static part of the grammar.

We would like to avoid using the GF compiler dynamically from the interactive knowledge base system, for the following reasons.

- Recompilation of grammar with new concepts might be required very often, for example after adding a definition. An actually dynamic vocabulary in the PGF runtime library seems more fitting.
- GF compiler is a heavier dependency for the engineering tool than the PGF library alone.
- GF compiler distribution has license incompatibility with RailCOMPLETE.

Hopefully, this functionality is covered by C runtime PGF library *proper noun callbacks*.

RailCOMPLETE (.NET) integration

Some consideration for integrating a GF CNL into a .NET desktop application without to many heavy dependencies.

1. We need to distribute our application in executable form. As we are not using Haskell in the RailCOMPLETE software, Haskell libraries would be a heavy dependency. We will focus instead on the C runtime, and assume that the C runtime has the same capabilities as the Haskell runtime. At least when disregarding higher-order and dependent types, this seems to be the case. The C runtime has been used in other projects with platform integrations.
2. The C runtime compiles with autotools and GCC. These are available on Windows through MinGW. I am having a bit of trouble to find out exactly what MinGW libraries must be distributed together with an application, and would prefer to also eliminate this dependency by using the Microsoft (MSVC) compiler. I managed to do a port the GF runtime to CMake and MSVC, which is not entirely trivial because (a) the code uses some GCC-specific extensions of C, and (b) the MSVC doesn't fully support the C99 standard, and (c) exporting functions and data from a library is slightly different on Windows. Some code changes were required, but the result was a very lightweight library (a few hundred kB). I can later clean this up and offer it to the GF developers, if they are willing to pollute the code with some MSVC-specific workarounds.
3. A .NET wrapper library is required to do a practical/idiomatic and safe (in terms of memory/resources) interface to the C library. The Python and Java wrappers show how to do this, but the Python wrapper, for example, is 3000 lines of C code, which could require a bit of effort to port. I have started on this, and it seems to be doable in a few days work, especially if some features can be (temporarily) omitted.
4. Manipulation of the AST and passing the AST from the library to the application code, is done through strings. To have type-safe manipulations, the GF compiler can generate Haskell code which transforms the strings into a Haskell algebraic data type. A wrapper library in another language should have something similar if it aims to manipulate a specific application grammar, which we want to do. Python (and Java?) wrappers lack this, it seems, and use instead a general data type for expressions. Maybe applications in these languages are more geared towards treating grammars in general, not specific application grammars.

Schedule

- **1 Nov - 14 Dec** Learn GF, make LL CNL, start on HL CNL, before Dec 14, when John takes parental leave.
- **1 Jan - 1 Feb** Work with Gerardo when he is back after new year, on the assumptions about logic in the verification (Datalog).
- **Feb 2017?** When CNL is well under way, start exploring
  - Regulations which need more than Datalog, e.g. SAT solving.
  - Synthesis/constraint solving for railway signalling design.
- **Mar 2017** Write paper for suitable publication venue. Gerardo will look for possibilities.

Literature list and links

CNL

- Thomas Kuhn survey
- Aarne Ranta, Computational semantics in type theory - Mathématiques et sciences
- Hähnle, Johannisson, Ranta, An Authoring Tool for Informal and Formal Requirements Specification
- Burke, Johannisson, Translating Formal Software Specifications to Natural Language
- Ranta: translating between logic and language
- Key Book chapter by Johannisson
- Bringert, Ranta: A pattern for almost compositional functions

Applications

- Emani: An approach for automatic formalization of business rules

GF

- Web page

  http://www.grammaticalframework.org/
- Library synopsis

  http://www.grammaticalframework.org/lib/doc/synopsis.html
- MOLTO application grammar building (“best practices”)

  http://www.molto-project.eu/sites/default/files/MOLTO_D2.3.pdf
- PhD thesis of Krasimir Angelov

Tools

- Minibar

  http://cloud.grammaticalframework.org/minibar/minibar.html
- Syntax editor (tree / linear)

  http://cloud.grammaticalframework.org/syntax-editor/editor.html
- CSE-ID splits sentences (shallow parsing)

  http://ad-wiki.informatik.uni-freiburg.de/research/Projects/CSDIE
- Illinois chunker also splits sentences (shallow parsing)

  https://cogcomp.cs.illinois.edu/page/software_view/Chunker
- Stanford parser gives complete parses

  http://nlp.stanford.org/

Other notes

Why don't we store the syntax tree instead of the source code?

http://softwareengineering.stackexchange.com/questions/119095/why-dont-we-store-the-syntax-tree-instead-of-the-s

- Requiring strict correspondence of AST and concrete syntax loses whitespace, comments, meaningful formatting, etc.
- Half-written code or almost-correct code is meaningful to the human reader, but the AST representation is not meaningful at all.
- No two languages are perfect matches.

Editor ideas

- Outer/inner editor: the regulations might require structuring in larger documents, while we would like to keep the CNL editor on a sentence-scale.

  See https://clearly.pl/tutorial/