An Ontology for Specifying and Parsing Knowledge Representation (KR) Structures and Notations

Philippe Martin (Philippe.Martin@univ-reunion.fr) [1] and
Jérémy BÉNARD (jeremy.benard@logicells.com) [2]

[1] Associate Professor, EA2525 LIM, University of La Réunion island, France
(+ adjunct researcher of the School of ICT, Griffith University, Australia)

[2] PhD student, GTH/Logicells + University of La Réunion island, France

Plan

1. Introduction - solving old parsing/export/translation problems
   => *"fully" representing* the involved languages (here KRLs)

2. Top level ontology - uppermost types of our ontology for KRLs (models + notations)

3. KRL model ontology - some subtypes of Abstract_phrase

4. Example - representation of a simple phrase and its abstract structure

5. Example - FL specification of the abstract parts of 2 very simple notations

6. Demo

7. Conclusions

1. Introduction - solving old parsing/export/translation problems => *"fully" representing* the involved languages (here KRLs)

Classic approaches:

* parser generators (e.g., since 1975, Lex&Yacc) + translation/export procedures/rules
* "structured document/editor/model" formating approaches, e.g., XML + XSLT + CSS + GRDDL
  or before 1996 (Thot, Centaur, ...): S/Typol + T + P
  or since 2005: RDF + Fresnel/SparqlTemplate/...
* XBNF (Botting, 2012): EBNF extention *towards* KR

=> exploitation of grammars+ASTs only, not KRs *of* KRLs

=> writing of one *syntactic* model (grammar/DTD/script/template) for each structure/presentation

+ lack of inferencing possibilities

=> - for *programmers*: importing, exporting or translating between KRLs is "difficult"

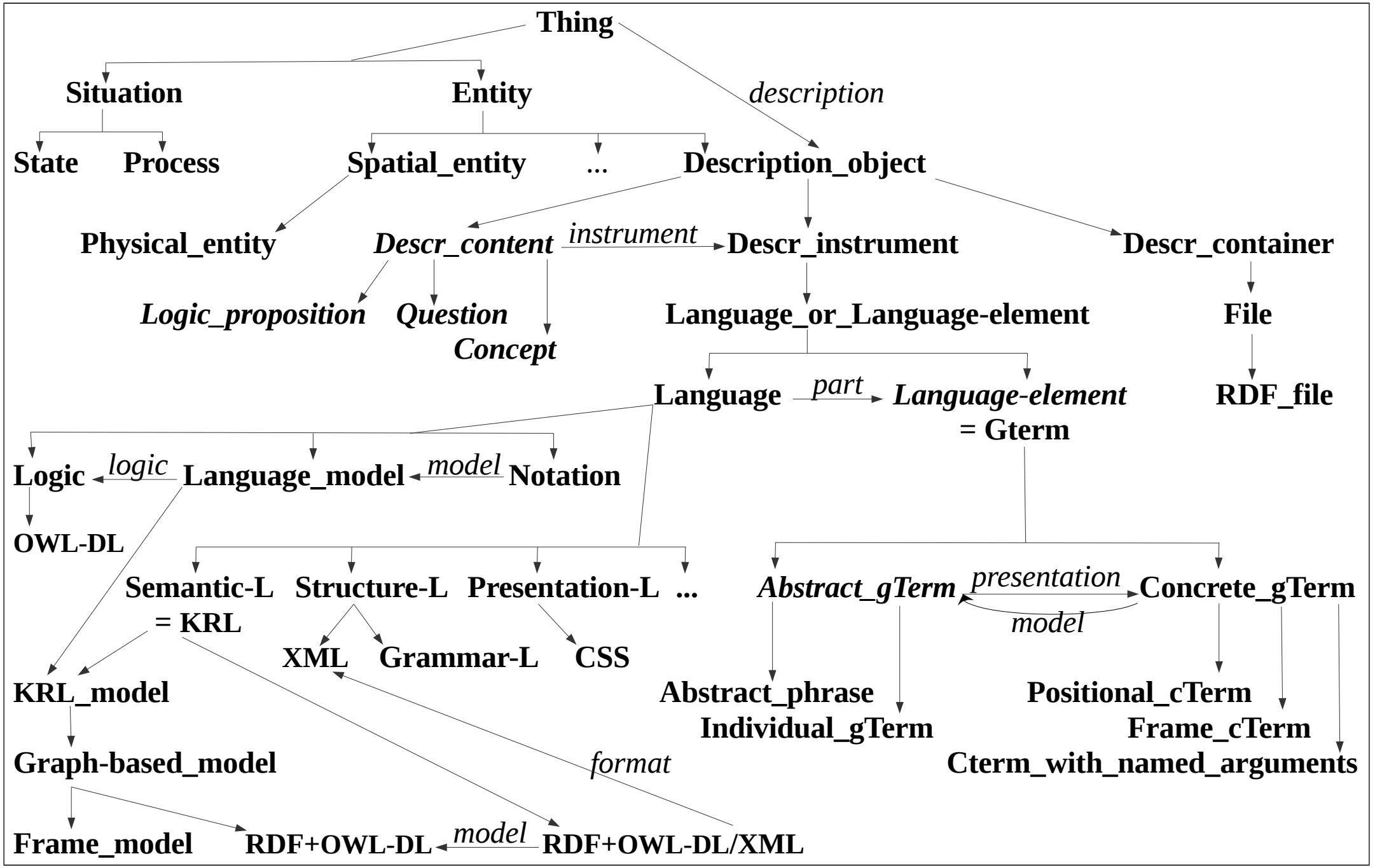
- for KRL *end-users*: adapting, extending or mixing notations is nearly impossible

=> knowledge sharing and re-use is reduced

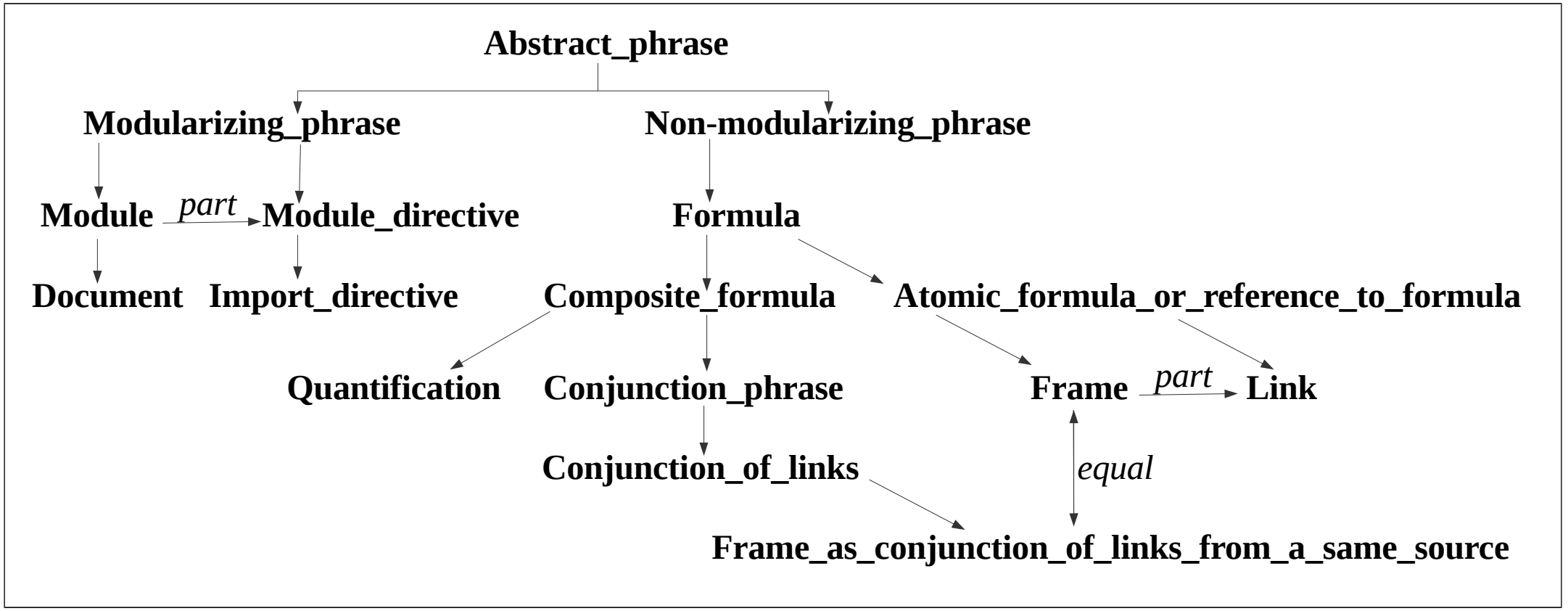
Fully ontology-based approach (=> use of "language ontologies"):

- 1 ontology of KRL models (-> extending existing ones) + 1 ontology of KRL notation+presentation *(new !)*
- letting each *end-user* specialize these ontologies to specify a new KRL (if he wishes to)
- 1 generic tool for parsing/exporting/translating from/to/between these specified KRLs

2. Top level ontology - the uppermost types of our ontology for KRLs (models + notations)


Thing
Situation
Entity
description
State
Process
Spatial_entity
...
Description_object
Physical_entity
Descr_content
instrument
Descr_instrument
Descr_container
Logic_proposition
Question
Concept
Language_or_Language-element
File
Language
part
Language-element
= Gterm
RDF_file
Logic
logic
Language_model
model
Notation
OWL-DL
Semantic-L
= KRL
Structure-L
Presentation-L
...
Abstract_gTerm
presentation
model
Concrete_gTerm
XML
Grammar-L
CSS
KRL_model
Abstract_phrase
Individual_gTerm
Positional_cTerm
Frame_cTerm
Cterm_with_named_arguments
Graph-based_model
format
Frame_model
RDF+OWL-DL
model
RDF+OWL-DL/XML

3. KRL model ontology - some subtypes of Abstract_phrase



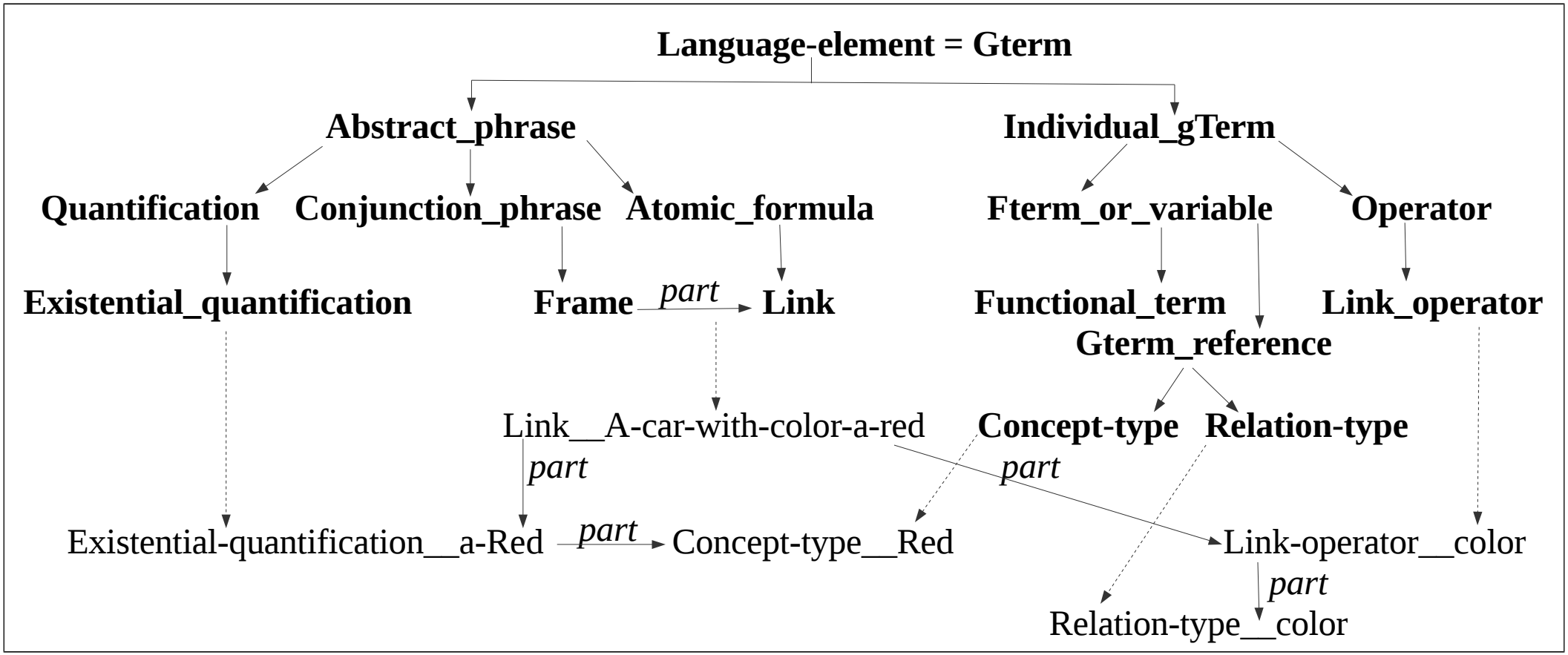
4. Example - representation of a simple phrase and its abstract structure

In English: "There exists a **car** which is **red** (one shade of red; it may have other shades or other colors)".

In Formalized-English: `a **Car** with *color* a **Red**´. In RIF-PS: Exists ?car ?red (*color*(?car#**Car** ?red#**Red**))

In FL: a **Car** *color :* a **Red** ; In RIF-PS: Exists ?car ?red ?car#**Car** [color -> ?red#**Red**].

In N-Triples: **Car8** *color* **Red3** . **Car8** *type* **Car** . **Red3** *type* **Red** .



5. Example - FL specification of the abstract parts of 2 very simple notations

N-triples = ^(**KRL** *r_only_such_part_of_that_type :* ^(**Phrase** > *Link*)
^(**Individual_gTerm** > **Constant_or_variable**));

JSON-LD *r_only_such_part_of_that_type :*
^(**Phrase** *rc_type :* **fc_list-like_infix-frame_type** _(.{**JSON-LD**},"","{",",","}"))
^(**Half_link** *rc_type :* **fc_half-link_type** _(.{**JSON-LD**},"",":","",""))
^(**Module_header** *rc_type :* **fc_list-like_infix-frame_type** _(.{**JSON-LD**},'"@context:"',"{",",","}"))
^(**Module_body** *rc_type :* **fc_list_type** _(.{**JSON-LD**},"","",",""))
^(**Formula** > ^(**Minimal_frame** *r_operator :* 1 **Constant_gTerm**)) //only 1 destination per link
^(**Fterm_or_variable** > **Constant_or_set_or_closed_list**)
^(**Set** *rc_type :* **fc_list_type** _(.{**JSON-LD**},"[",",","]"))
^(**Closed_list** > ^(**Frame** *r_part :* 1 .[**r_container**, **Closed_list**], //1st way to represent a list in JSON-LD
rc_type : **fc_half-link_type** _(.{**JSON-LD**},"","@container",":","@list",""))
^(**Frame** *r_part :* .[**r_list**, 1 **Set**],
rc_type : **fc_half-link_type** _(.{**JSON-LD**},"","@list",":","",""))); //2nd way

6. Demo

7. Conclusions

The examples focused on "abstract terms" but specifying "concrete terms" is similar.
Specifying grammars (instead of KRLs) is also similar.

- Less lines to write for specifying a KRL model+presentation than to write its grammar
- No parsing/translation/export tool/schema to write in addition

=> Much simpler and much more powerful: - end-users can specify their own KRLs
- models/notations/KRLs can be compared

=> a much better alternative to XML as a meta-language
and XML+XSLT+CSS can be re-used for presentation purposes.

Given the specification of a target KRL, generating knowledge in this KRL has been implemented.
Allowing the use of a presentation language (e.g., HTML or XML+XSLT+CSS) for specifying the presentation (e.g., in bold) of particular language elements has not yet been implemented.
Given the specification of a source KRL, parsing is currently done in an ad-hoc way
and the generation of parsing rule in a given grammar has not yet been implemented.