Knowledge representation in CGLF, CGIF, KIF, Frame-CG and Formalized-English

Philippe Martin

Distributed System Technology Centre
Griffith University - PMB 50 Gold Coast MC, QLD 9726 Australia
philippe.martin@gu.edu.au

Abstract. This article shows how CGLF, CGIF, KIF, Formalized-English and Frame-CG can be used in a panorama of knowledge representation cases. It highlights various inadequacies of CGLF and CGIF, advantages provided by high-level expressive notations, and the KIF translations provide a logical interpretation. Knowledge providers may see this document as a guide for knowledge representation. Developers may see it as a list of cases to take into account for their notations and inferences engines.

1 Introduction

A knowledge-based system (KBS) generally uses only one *model* to store and exploit knowledge, e.g. a semantic network model such as Conceptual Graphs (CGs) [1], but may import/export (i.e. accept/present) representations in various *notations*, e.g. CGDF (CG Display Form), CGLF (CG Linear Form), CGIF (CG Interchange Format) and KIF (Knowledge Interchange Format) [2].

Inference engines may or may not exploit all parts of the stored knowledge. E.g. knowledge retrieval engines may be efficient and provide interesting results by considering all contexts as positive contexts and ignoring the various logical meanings of collection types. The more the model has features, the more *inferencing* can be done. The more the notations are expressive, the more information can be entered precisely, exploited, presented and exchanged. In this view, the issues of computability and decidability are *not* related to notations or models (provided they are not restricting) but to inference engines: with expressive notations, each engine developer may decide which features to exploit in order to deal with the problems of efficiency, consistency and completeness. Thus, restricting notations can never be an advantage, they can only limit and bias knowledge modelling and the many possible inferencing techniques. This article does not explore the computability of inferencing operations made possible by expressive notations; this vast subject is the focus of the description logics litterature.

Are some notations better than others for knowledge representation and exchange? According to the above, expressiveness is one criteria. Readability and conciseness are important too, since they ease the understanding of knowledge and, for the developers, ease debugging. Knowledge entering easiness is a criteria related to conciseness and how high-level the notation is (i.e. how many ontological distinctions have a special intuitive syntax). Also related and important is the knowledge normalizing effect of the notation: the fewer choices a knowledge provider has for representing a piece of information (in a simple way), the easier it is to develop an inference engine (or knowledge matching technique) that can relate and compare this representation to other ones. Thus, high-level expressive notations seem better for knowledge representation and exchange than low-level expressive notations such as KIF, or low-level restricted notations such as RDF/XML.

From a knowledge provider's viewpoint, another problem (with all notations but especially low-level or restricted ones), is how to express knowledge. The documentations about notations often only provides a grammar, a few simple examples, and omit to explain how to represent more complex cases commonly found in natural language sentences, or to state that some of these cases cannot be represented. This lack of details also make notations difficult to compare. For example, the documentations of CGIF [1] and RDF/XML [3] are currently very poor. The documentation of KIF is completed by the Ontolingua library[1] but several knowledge representation cases are still difficult to find. Knowledge representation with CGLF is relatively well documented in [8] and [9] but CGLF is not standardized and inconsistent usages are often encountered, even in Sowa's descriptions [8] [9]. Finally, the logical interpretation of many keywords (syntactic sugar for some features) is not always provided, as for example is the case for CGIF and RDF/XML.

To provide some answers to the previous questions and problems, this article presents a panorama of knowledge representation features and shows how various notations can be used (or extended to be used) to cover these features. (We focused on features that are commonly required to represent of natural language sentences and knowledge representation in general, e.g. numerical quantifiers, but rarely or badly handled by most notations; [8] and [9] were initially used as models). In addition to CGLF, CGIF and KIF, this document presents two notations derived from CGLF and designed to be as intuitive[2] as possible in all the presented cases: Formalized English (FE) and Frame-CG (FCG) [6] [3]. (RDF/XML has also been examined[4] but because of space limits, the results are not presented here)[5].

Knowledge providers may see this document as a guide for knowledge representation. KBS developers may see it as a list of cases to take into account. Language developers may see it as a workbench for comparing their notations to others. The translation into KIF also provides a logical interpretation for the other notations.

In each example of this article, we follow the lexical conventions (e.g. singular nouns, English spelling) and ontological conventions that we advocated in [6] for knowledge comparison, retrieval and exchange. Except in Section 11 (which deals with category declaration) the categories are supposed to be already declared.

[1] http://www-ksl-svc.stanford.edu:5915/

[2] The use of English articles or expressions as (extended) quantifiers, one of our ideas to obtain a more intuitive and "knowledge normalizing" notation, was also applied (although to a less extent) in KM, the Knowledge Machine notation [4].

[3] WebKB-1, our first Web-based KBS, imports and exports CGLF, FCG and FE. WebKB-2 [7] currently only uses FCG and partially exports in RDF/XML, but will later also import and export in FE, CGLF, CGIF and KIF. Grammars and parsing examples of these notations are at: http://www.webkb.org/doc/grammars/

[4] See http://www.webkb.org/doc/translations.html

[5] This article also *only* presents a panorama of "logical features"; ontological examples in FCG can be found on the WebKB site: http://www.webkb.org

2 Conjunctive Existentially Quantified Statements

Here is an example of such simple forms of knowledge. "E" is for "English".

```
E:    Tom owns a dog that is not Snoopy.
FE:   Tom is owner of a dog different_from Snoopy.
FCG:  [Tom, owner of: (a dog != Snoopy)]
CGLF: [T:Tom]<-(owner)<-[dog:*x!=Snoopy]  //T: uppermost concept type
CGIF: [dog:*x] (owner ?x Tom) (different_from ?x Snoopy)
KIF:  (exists ((?x dog)) (and (owner ?x Tom) (/= ?x Snoopy)))
```

A problem is that `different_from` is not in the CG standard [1], thus leading people to use other identifiers for this basic form of negation and hence making knowledge matching difficult. As long as the CG standard, or an ontology *referred by* the CG standard, does not introduce a number of keywords for common relations and quantification, knowledge sharing and inferencing with CG will remain difficult.

The CGLF statement uses "!=" in the declaration of a coreference variable to specify that the variable does not refer to a certain individual. This syntax has often been used, e.g. by Sowa [8] [9], but is not included in the *minimal* CGLF grammar given in the CG standard [1]. We have used it because it is convenient and semantically equivalent to what is expressed with the other notations.

3 Contextualization

Contexts allow us to represent statements over statements. Contexts are often represented via delimitors, e.g. '...' in FE, [...] in FCG, '(...) and ^(...) in KIF.

```
E:   Tom believes Mary now likes him (in 2002) and before she did not.
FE:  Tom is believer of ' *p 'Mary is liking Tom' at time 2002'
     and is believer of '!*p is before 2002'.
FCG: [Tom,believer of: [*p [Mary, agent of:(a liking,object:Tom)],time:2002],
          believer of: [!*p, before: 2002] ]
CGLF:[proposition: *p [T:Mary]<-(agent)<-[liking]->(object)->[T:Tom] ]
     [T:Tom]- { (believer)<-[ [situation: ?p]->(time)->[time:"2002"],
                (believer)<-[ [situation:~?p]->(before)->[time:"2002"] ] }
CGIF:[proposition *p: (agent [liking *l] Mary)  (object ?l Tom) ]
     (believer [situation: (time [situation: ?p] "2002")] Tom)
     (believer [situation: (before [situation: ~[?p]] "2002")] Tom)
KIF: (exists (?p)
      (and (= ?p '(exists((?x liking)) (and (agent *l Mary)(object ?l Tom))))
           (believer ^(time ,?p 2002) Tom) //',?p'->the value of ?p is quoted
           (believer ^(before (not ,?p) 2002) Tom)))
```

Relations of type `believer`, `time` and `before` connect an instance of the type `situation` to another object. In CG, it is customary to distinguish the "proposition" stated by a statement/graph/formula from the described "situation". This distinction is explicit in CGLF and CGIF above. However, making this distinction is sometimes difficult for novices, and it is inconvenient because it leads to adding several intermediary contexts. Since these contexts can be automatically inserted by a parser according to the signatures of the used relations, we have not included the intermediary contexts in the other notations. (We also assumed parsers can understand that 2002 is a date, based on relation signatures).

In CGLF and CGIF, a coreference variable is introduced with the prefix '*' and re-used within the same context with the prefix '?'. Thus, two embedded sibling contexts may introduce variables with the same name but not referring to the same object. We do not think this approach is easy to follow.

Instead, in FE and FCG, variables may be prefixed by '?' or '*' (or '@' for collections, as in KIF) and a variable introduction is most often distinguished by being used *with* a type and a quantifier (hence, a variable introduction must precede its re-use). When, within a graph, a variable re-use exists in a context (c1) different from the context (c2) where the variable has been introduced, the convention is that the variable is assumed to have been introduced in the minimum upper context embedding c1 and c2 (in CGLF and CGIF, this has to be done explicitly by the user but this can be cumbersome and counter-intuitive).

Finally, FE and FCG also permit the introduction of *free variables* with the prefix '^'. Their semantics are the same as in KIF: within statements (as opposed to queries), these variables are assumed to be introduced with an universal quantifier in some upper context (as before, the lowest context that includes all the introductions and re-uses of the variables).

4 Universal Quantification

```
E:    Animals have exactly one head.
FE:   Any animal has for part 1 head.
FCG:  [any animal, part: 1 head]
CGLF: [animal: @forall]->(part)<-[head: @1]
CGIF: (part [animal: @forall] [head: @1])
KIF:  (forall ((?a animal)) (exists1 '?h (and (head '?h) (part ?a '?h))))
```

Here is our KIF definition of `exists1`:

```
(defrelation exists1 (?var ?predicate) :=
  (truth ^(exists (,?var) (and (,?predicate ,?var)
                 (forall (?y) (=> (,?predicate ?y) (= ,?var ?y)))))))
```

Problem with the CGLF and CGIF statements: `@1` is common but not standard.

5 Lambda Abstraction, Percentage, Possibility, Valuation

```
E:   At least 93% of healthy birds can fly.
FE:  At least 93% of [bird with chrc a good health] can be agent of a flight.
FCG: [at least 93% of (bird, chrc: a good health), can be agent of: a flight]
CGLF:[physical_possibility:
       [lambda(b)[bird:*b]->(chrc)->[health]->(measure)->[value:good]: @>93%]
       <-(agent)<-[flight] ]
CGIF:[(lambda(bird *b) [health *h] (chrc ?b ?h) (measure ?h good)) *x: @>93%
         [physical_possibility: (agent [flight] *x)] ]
KIF: (defrelation healthy_bird (?b) :=
       (exists ((?h health)) (and (bird ?b) (chrc ?b ?h) (measure ?h good))))
     (forAtLeastNpercent 93 '?x healthy_bird
        (exists ((?f flight)) (physical_possibility (agent ?f '?x))))
```

Here is our KIF definition of forAtLeastNpercent (and associate functions):

```
(defrelation forAtLeastNpercent (?n ?var ?type ?predicate) :=
  (exists ((?s set))
   (and(truth ^(forall (,?var) (=> (member ,?var ,?s) (,?type ,?var)))
       (>= (numMembersSuchThat ,?s ,?predicate) (/ (* (size ,?s) ?n) 100)))))


(define-function numMembersSuchThat (?set ?p) :-> ?num :=
  (if (and (set ?set) (predicate ?p)) (numElemsSuchThat (listOf ?set) ?p)))

(define-function numElemsSuchThat (?list ?p) :-> ?num :=
  (cond ((null ?list) 0)
        ((list ?list) (if ?p (1+ (numElemsSuchThat (rest ?list) ?p))))))
```

The CGLF and CGIF representations have several problems.

First, although @>93 is permitted as "defined quantifier" by the CG standard, @>93% is syntactically incorrect. Furthermore, since a "defined quantifier" can be anything and cannot actually be defined, its meaning is left implicit (the standardization of common extended quantifiers such as @>93% is necessary).

Second, physical_possibility is not in the current CG standard.

Third, only the agent relation should be contextualized. In CGLF, this is cumbersome. In CGIF, should the concept with type physical_possibility be in the referent part of the concept with the numerical quantifier (@>93%)? What is the actual meaning of this construction? How can the scope of this quantifier be delimited otherwise? The CG standard says that "for complex mixtures of quantifiers, the scope can be delimited by transparent contexts (marked by context brackets [ ] with no type label)". But is it consistent with the other uses of concept embedding?

Fourth, good is not in the CG standard. FE and FCG have five keywords for quantitative valuation: good, bad, important, small, big, great. This allows the user to avoid introducing adjectives (categories with adjectives as names) into the ontology and hence makes it more (re-)usable [6]. We do not believe that average users can or should define valuations for each possible measurable quantity (e.g. what would good_boy, good_work, good_food and bad_food mean?).

Fifth, measure and value are not standard either. Extensions or ontological conventions are needed to permit knowledge exchange and exploitation.

Sixth, in the CGIF statement, should *b and *x be merged into a single variable? The CG standard does not give indications.

Seventh, how to represent lambda-abstractions in CGLF? Sowa put them in referent fields of concepts, and used the character λ in his articles and the HTML encoding of this character (λ) in the CG standard (and sometimes even λ₁ and λ₂). We adopted a more classic and consistent notation closer to the one used in CGIF.

We have not found a simple way to represent a lambda-abstraction (that is, an *anonymous* type declaration) in KIF. Hence, we used a normal type declaration.

The above example can be modified to refer to "most birds" instead of "93% of birds". In FE and FCG, the keyword most may be used and is is equivalent to at least 60% (hence, it can be translated to KIF in this form). In CGLF and CGIF, @most may be used but its meaning has not been made explicit.

6 Negations, Exclusions and Alternatives

We have already seen two forms of negation: the `different_from` relation (`/=` in KIF), and the negation of a statement ("not" in KIF) which is more difficult to exploit by inference engines and leaves room for ambiguity. For example, "Tom does not own a blue car" may mean that "Tom has a car but not blue" or "Tom does not have a car". Thus, it is better to use the first form, or break statements into smaller blocks connected by coreference variables to reduce or avoid ambiguities.

Here is a variant of the first form: negation on types.

```
E:   Tom owns something that is not a car.
FE:  Tom is owner of a !car.               FCG:   [Tom, owner of: a !car]
CGLF:[T:Tom]<-(owner)<-[~car:*]            CGIF:  (owner [~car] Tom)
KIF: (exists (?type ?x) (and (owner ?x Tom) (holds ?type ?x) (/= ?type car)))
```

Exclusion between objects (and hence, some forms of negation) may also be represented via collections of exclusive objects. FE and FCG use OR-collections and XOR-collections as syntactic means to store "or" and "xor" relations between objects (types, instances or statements). Here is an example of OR-collection between instances. (Note: `red`, `yellow` and `orange` are not instance but subtype of `color`, and have many subtypes, e.g. `crimson, dark_red` and `chrome_red`. Their instances are the actual occurrences of color that physical objects have.)

```
E:   Tom's car is red, yellow or orange.
FE:  Tom is owner of a car that has for color OR{a red, a yellow, an orange}.
FCG: [Tom, owner of: (a car, color: OR{a red, a yellow, an orange})]
CGLF:[Tom]<-(owner)<-[car]->(chrc)->[color]->(kind)->[TYPE:OR{red,yellow,orange}]
CGIF:[car:*x] (owner *x Tom) (color *x [red|yellow|orange:])
KIF: (exists ((?x car) ?c)
      (and (owner ?x Tom) (color ?x ?c) (or (red ?c)(yellow ?c)(orange ?c))))
```

There is no usual way to represent OR collections in CGLF; we used the FCG way rather than the CGIF way because it is more general (in CGIF, only types can be OR-ed without introducing contexts).

In this example, it would have been simpler to use a type such as `warm_color` instead of the OR-collection of `red`, `yellow` and `orange` (and this form makes inferencing easier). More generally, this section shows that a negation can be represented in numerous ways and that these representations are difficult for an inference engine to compare and hence exploit fully. Both for knowledge exchange with frame-based systems and for knowledge inferencing, `different_from` relations between instances or types should be prefered to other forms of negations.

7 Collections and Quantifier Precedence

Collections have been introduced in the previous section and via examples using numerical quantifiers. In this section, we show how various interpretations of the English sentence *"4 judges have approved 3 laws"* (and some variations of it) can be interpreted. By studying how to represent relations between members of two simple collections, we illustrate the importance of specifying how a collection must be interpreted, and show how to handle complex cases of quantifier precedence (between numerical, existential and universal quantifiers).

The sentence *"4 judges have approved 3 laws"* is ambiguous. The 4 judges may have individually or collectively approved 3 laws (the same 3 or not), and "collectively" may have two meanings: the participation in a "unique" approval act or the approval of "most" of the laws (or a combination of both as illustrated in the last example of this section). In this paper, "judges acting together/collectively" means that "there exists an act and each of the judges is an agent of that act". This interpretation of "collectiveness" was used by Sowa in [8] and implies that the act can only be represented by a concept node, not by a relation node (this has not been made explicit by Sowa).

In CGs [8], any collection in a concept of a CG can be specified as having a *distributive interpretation* (each member of the collection individually participates to the relations associated with the concept), a *collective interpretation* (the members collectively participate in the relations associated with the concept), a *default interpretation* (an unspecified mix of collective and distributive interpretation) or a *cumulative interpretation* (the relations are about the collection itself). (The current CG standard does not specify what the various interpretations of a collection can be, not even within the CGIF grammar; it mentions the keyword `Col` as a "collective designator" but not the keywords `Dist` and `Cum` used in [8]).

The first example keeps the ambiguity of the above sentence (both collections have the default interpretation). The 's' at the end of `judges` and `laws` in the FE and FCG representations are supposed to be automatically removed (as does WebKB-2 when a numerical or universal quantifier is involved). To highlight the logical interpretations, this section provides predicate logic (PL) translations instead of FE translations.

```
E:    4 judges have (each/together) approved 3 laws.
FCG:  [4 judges, agent of: (an approval, object: 3 laws)]
CGLF: [judge:{*}@4 @certain]<-(agent)<-[approval]->(object)->[law:{*}@3]
CGIF: (agent [approval:*a] [judge: @4 @certain]) (object ?a [law: {}@3])
KIF:  (forAllN 4 '?j judge (forAllN 3 '?l law
          (exists ((?a approval)) (and (agent ?a '?j) (object ?a '?l)))))
```

PL: $\exists js\ set(js) \wedge size(js, 4) \wedge \forall j \in js\ \ \exists ls\ set(ls) \wedge size(ls, 3) \wedge \forall l \in ls$
$\exists a\ approval(a) \wedge agent(a, j) \wedge object(a, l)$

For modularity, we introduced the "quantifier" `forAllN`.

```
(defrelation forAllN (?num ?var ?type ?predicate) :=
  (exists ((?s set)) (and (size ?s ?num)
     (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                 (and (,?type ,?var) ,?predicate)))))))
```

In CGLF and CGIF, the order of the quantifiers at a same level in a context is specified via a simple convention: the existential quantifier marked by the keyword `@certain` has precedence over the universal/numerical quantifiers which have precedence over the over existential quantifiers. Remaining ambiguities have to be solved by the user via the addition of contexts. In FE and FCG, the order and scope of the quantifiers follow the order and structure of the graphs. The next example shows a simple inversion of the quantifier scopes.

```
E:    3 laws have been approved by 4 judges (each/together).
FCG:  [3 laws, object of: (an approval, agent: 4 judges)]
CGLF: [judge:{*}@4]<-(agent)<-[approval]->(object)->[law:{*}@3 @certain]
CGIF: (agent [approval:*a] [judge:@4]) (object ?a [law: {}@3 @certain])
KIF:  (forAllN 3 '?l law (forAllN 4 '?j judge
         (exists ((?a approval)) (and (agent ?a '?j) (object ?a '?l)))))
```

PL: $\exists ls\ set(ls) \wedge size(ls,3) \wedge \forall l \in ls\ \ \exists js\ set(js) \wedge size(js,4) \wedge \forall j \in js$
$\exists a\ approval(a) \wedge agent(a,j) \wedge object(a,l)$

In FE and FCG, the *collective interpretation* is specified via the keywords `set of`, `group of`, `together`, `bag of`, `list of`, `sequence of` or `alternatives` (the first three are synonyms; in this paper, we most often use `set`). In CGLF and CGIF, the keyword `Col` is used, and the collection is assumed to be a set.

If we take the two previous examples and gradually introduce the collective interpretation for the collections, we obtain five different logical interpretations (instead of six because when both collections are collectively interpreted, the inversion of quantifier scopes does not change the meaning). Below are three of these combinations (the other two are: "A group of 3 laws has been approved by 4 judges" and "A group of 4 judges has approved 3 laws"). In the third case, we give the three equivalent FCG statements.

```
E:    4 judges have (each/together) approved a group of 3 laws.
FCG:  [4 judges, agent of: (an approval, object: a set of 3 laws)]
CGLF: [judge:{*}@4 @certain]<-(agent)<-[approval]->(object)->[law:Col{*}@3]
CGIF: (agent [approval:*a] [judge:@4]) (object ?a [law:@Col{}@3 @certain])
KIF:  (forAllN 4 '?j judge (exists ((?ls set) (?a approval))
        (forAllIn ?ls 3 '?l law (and (agent ?a '?j) (object ?a '?l)))))
```

PL: $\exists js\ set(js) \wedge size(js,4) \wedge \forall j \in js\ \ \exists ls\ set(ls) \wedge size(ls,3) \wedge$
$\exists a\ approval(a)\ \forall l \in ls\ agent(a,j) \wedge object(a,l)$

```
E:    3 laws have been approved by a group of 4 judges.
FCG:  [3 laws, object of: (an approval, agent: a set of 4 judges)]
CGLF: [judge:Col{*}@4]<-(agent)<-[approval]->(object)->[law:{*}@3 @certain]
CGIF: (agent [approval:*a] [judge:@Col{}@4]) (object ?a [law:@3{} @certain])
KIF:  (forAllN 3 '?l law (exists ((?js set) (?a approval))
        (forAllIn ?js 4 '?j judge (and (agent ?a '?j) (object ?a '?l)))))
```

PL: $\exists ls\ set(ls) \wedge size(ls,3) \wedge \forall l \in ls\ \ \exists js\ set(js) \wedge size(js,4) \wedge$
$\exists a\ approval(a)\ \forall j \in js\ agent(a,j) \wedge object(a,l)$

```
E:    A group of 4 judges has approved a group of 3 laws.
FCG:  [a set of 4 judges, agent of: (an approval,object:a set of 3 laws)]
 or:  [a set of 3 laws, object of: (an approval,agent:a set of 4 judges)]
 or:  [an approval, agent: a set of 4 judges, object: a set of 3 laws]
CGLF: [judge: Col{*}@4]<-(agent)<-[approval]->(object)->[law: Col{*}@3]
CGIF: (agent [approval: *a] [judge: @Col{}@4])
        (object ?a [law: @Col{}@3 @certain])
KIF:  (exists ((?a approval) (?js set) (?ls set))
        (forAllIn ?js 4 '?j judge (forAllIn ?ls 3 '?l law
          (and (agent ?a '?j) (object ?a '?l)))))
```

PL: $\exists a\ approval(a) \wedge \exists js\ set(js) \wedge size(js,4) \wedge\ \exists ls\ set(ls) \wedge size(ls,3) \wedge$
$\forall j \in js\ \forall l \in ls\ agent(a,j) \wedge object(a,l)$

Here is how we define the "quantifier" `forAllIn`.

```
(defrelation forAllIn (?s ?num ?var ?type ?predicate) :=
  (and (size ?s ?num)
       (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                   (and (,?type ,?var) ,?predicate))))))
```

In FE and FCG, the *distributive interpretation* is specified via the keyword `each`. In CGLF, the keyword `Dist` is used. The CG standard does not address this issue but allows `@Dist` in CGIF. If we introduce the distributive interpretation into the previous seven combinations, we obtain nine different logical interpretations. Here are two of them.

```
E:    4 judges have each approved 3 laws.
FCG:  [each of 4 judges, agent of: (an approval, object: 3 laws)]
CGLF: [judge: Dist{*}@4]<-(agent)<-[approval]->(object)->[law:{*}@3]
CGIF: (agent [approval:*a] [judge: @Dist{}@4]) (object ?a [law:{}@3])
KIF:  (forAllN 4 '?j judge (exists!! '?j '?ls set (forAllIn '?ls 3 '?l law
        (exists!! '?j '?a approval (and (agent '?a '?j)(object '?a '?l))))))
```

PL: $\exists js\ set(js) \wedge size(js,4) \wedge \forall j \in js\ \ \exists!!ls\ set(ls) \wedge size(ls,3) \wedge$
$\forall l \in ls\ \exists!!a\ approval(a) \wedge agent(a,j) \wedge object(a,l)$

```
E:    4 judges have each approved a group of 3 laws.
FCG:  [each of 4 judges, agent of: (an approval,object: a set of 3 laws)]
CGLF: [judge:Dist{*}@4]<-(agent)<-[approval]->(object)->[law:Col{*}@3]
CGIF: (agent [approval:*a] [judge:@Dist{}@4]) (object ?a [law:@Col{}@3])
KIF:  (forAllN 4 '?j judge (exists!! '?j '?ls set (exists!! '?j '?a approval
        (forAllIn '?ls 3 '?l law (and (agent '?a '?j) (object '?a '?l))))))
```

PL: $\exists js\ set(js) \wedge size(js,4) \wedge \forall j \in js\ \ \exists!!ls\ set(ls) \wedge size(ls,3) \wedge$
$\exists!!a\ approval(a)\ \forall l \in ls\ agent(a,j) \wedge object(a,l)$

Below is our KIF definition of `exists!!` ($\exists!!$). This quantifier permits us to specify that the judges are agent of different approvals and different laws (first example above) or groups of laws (second example above).

```
(defrelation exists!! (?var1 ?var2 ?type ?predicate) :=
    (truth ^(exists (,?var2)
       (and (,?type ,?var2) (,?predicate ,?var1 ,?var2)
            (forall (?x) (=> (,?predicate ,?var1 ?x) (= ,?var2 ?x)))
            (forall (?y) (=> (,?predicate ?y ,?var2) (= ,?var1 ?y)))))))
```

Finally, we can introduce "most" as an interpretation of collectiveness in the previous (7+9=16) combinations. Hence, 16 new logical interpretations. Here is one.

```
E:    A group of 3 laws has been approved by most in a group of 4 judges.
FCG:  [a group of 4 judges, agent of:
           (an approval, object: most in a group of 3 laws)]
 or:  [most in a group of 3 laws, object of:
           (an approval, agent: a group of 4 judges)]
CGLF: [judge:Col{*}@4]<-(agent)<-[approval]->(object)->[law:Col{*}@3 @most]
CGIF: (agent [approval:*a] [judge:@Col{}@4])(object ?a [law:@Col{}@3 @most])
KIF:  (exists ((?a approval) (?js set) (?ls set))
        (forAllIn ?js 4 '?j judge (forMostIn ?ls 3 '?l law
          (and (agent ?a '?j) (object ?a '?l)))))
```

PL: $\exists a\ approval(a)\ \wedge\ \exists js\ set(js) \wedge size(js,4)\ \wedge\ \ \exists ls\ set(ls) \wedge size(ls,3)\ \wedge$
$\forall j \in js\ agent(a,j)\ \wedge\ \exists mostOfls\ set(mostOfls)$
$(\forall l \in ls\ (object(a,l) => l \in mostOfls))\ \wedge\ size(mostOfls) >= 2$
$//\ \ >= 2\ since\ size(ls)/2 = 1.5$

Here is how we define `forMostIn` (see Section 5 for `numMembersSuchThat`).

```
(defrelation forMostIn (?set ?num ?var ?type ?predicate) :=
  (and (size ?set ?num)
       (truth ^(forall (,?var) (=> (member ,?var ,?set) (,?type ,?var))))
       (>= (numMembersSuchThat ,?set ,?predicate) (* (size ,?set) 0.60))))
```

8 Intervals

```
E:   Tom has been running for 45 minutes to an hour.
FE:  Tom is agent of a run with duration a period with part 45 to 60 minutes.
FCG: [Tom, agent of: (a run, duration: (a period, part: 45 to 60 minutes))]
CGLF:[run]- { (agent)->[Tom],
              (duration)->[period]->(part)->[minute: Col{*}@45-60] }
CGIF:(agent [run *r] Tom) (duration ?r [period *d])
                          (part ?d [minute: @Col{}@45-60])
KIF: (exists ((?r run) (?p period) (?minutes set))
       (and (agent ?r Tom) (duration ?r ?p)
            (forAllInBetween ?minutes 45 60 '?m minute (part ?p '?m))))
```

Here is how we define `forAllInBetween`.

```
defrelation forAllInBetween (?s ?n1 ?n2 ?var ?type ?predicate) :=
  (exists (?n) (and (size ?s ?n) (>= ?n ?n1) (=< ?n ?n2)
    (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                (and (,?type ,?var) ,?predicate))))))
```

In these CGLF and CGIF, the collective interpretation is specified for the minutes so that the numerical quantifier has the lowest precedence. In FE and FCG, the graph structure is sufficient to specify the scopes of the quantifiers.

In all these notations, a concept of type `period` had to be introduced since the minutes participate in the same period/duration. This is the same problem as for the collective participation to an act: the act cannot be represented as a relation. Here, a relation of type `duration` cannot directly connect the run to the minutes. We only became aware of this problem when trying to produce the KIF representation.

9 Function Calls and Lists

Special syntactic sugar to distinguish functional relations from other relations is not mandatory since this distinction can be specified in the relation type declaration (hence, all notations permit function calls even if they do not permit function definitions). However, a syntactical difference eases readablility and syntactic checking. The next example involves two functions (`length`, `+`) and one relation ($<$).

```
E:    The length of the list "Tom, Joe, Jack" plus 1 is less than 5.
FE:   length(LIST{Tom,Joe,Jack}) + 1 < 5.
FCG:  [length(LIST{Tom,Joe,Jack}) + 1 < 5]
```

```
CGLF: [number:*x]<-<plus>- <-1- [number]<-<length><-[T: <Tom,Joe,Jack>],
                           <-2- [number:1]
      [?x]->(superior)->[number:5]
CGIF: <length [T: <Tom,Joe,Jack>] *l>  <plus ?l 1 [number]>
KIF:  (superior (+ (length (listof Tom Joe Jack)) 1) 5)
```

Problem with the CGIF notation: the CG standard specifies that angular brackets should be used to delimit lists but the CGIF grammar only accepts curly brackets. Furthermore, `length` and `plus` are not in the CG standard, thus leading people to use other identifiers and hence making knowledge comparison difficult.

In FE and FCG, the notation for functional relations can also be used to represent non-binary relations (in CGLF, CGIF and KIF, binary and non-binary relations have a similar syntax; this does not lead the knowledge provider to use binary relations only, and hence leads to less explicit and comparable statements [6]).

10 Higher-order Statements

First-order statements quantify over individuals. Higher-order statements also quantify over types. For example, describing the transitivity of a particular relation (e.g. "the part of a part is also a part") can be a first-order statement, but describing in general what a transitive relation is, requires a second-order statement. Since definitions will be presented in the next section, the next example does not define a type such as `transitive_binary_relation` but uses the characteristic `transitivity`.

```
E:    If a binary relation type rt is transitive
      then if x is connected to y by a relation of type rt, and
              y is connected to z by a relation of type rt,
           then x is connected to z by a relation of type rt.
FE:   If ‘a binaryRelationType *rt has for chrc the transitivity’
      then ‘if ‘^x has for *rt ^y that has for *rt ^z’
            then ‘^x has for *rt ^z’ ’.  //rt,x,y,z are free variables
FCG:  [ [a binaryRelationType *rt, chrc: the transitivity] =>
        [ [^x, *rt: (^y, *rt: ^z)] => [^x, *rt: ^z]
       ]]
CGLF: [IF: [binaryRelationType: *rt]->(chrc)->[transitivity]
       [THEN: [IF: [T: *x]->(&rt)->[T: *y]->(&rt)->[T: *z]
               [THEN: [?x]->(&rt)->[?z]
               ]]]]
CGIF: [IF: (chrc [binaryRelationType *rt] [transitivity])
       [THEN: [IF: (holds ?rt [T:*x] [T:*y]) (holds ?rt ?y [T:*z])
               [THEN: (holds ?rt ?x ?y)
               ]]]]
KIF:  (exists ((?t transitivity))
       (forall ((?rt binaryRelationType) ?x ?y ?z)
        (=> (chrc ?rt ?t)
           (=> (and (holds ?rt ?x ?y)(holds ?rt ?y ?z)) (holds ?rt ?x ?z)))))
```

In CGLF, we used '&' to specify the mapping from the relation type `rt` to a free variable referring to a relation of type `rt`. [Sowa, 1993] uses the greek character ρ but this character is not easy to enter. An alternative would be to keep the variable re-use prefix '?' since the location of the re-use (i.e. within a relation) seems sufficient to highlight the special semantic. We adopted this second solution in FE and FCG (in the example, '^' is used instead of '?' or '*' when a free variable is used). In CGIF, since the current syntax does not permit variables for relation types, we used a universal quantifier and the relation type `holds`, as in KIF.

11 Declarations and Definitions

In RDF/XML, a category is uniquely identified by a URI, e.g. `http://www.foo.com` and `http://www.bar.com/doc.html#car`. In a multi-user KBS such as WebKB-2 [7], user identifiers are more convenient knowledge source identifiers than document URIs. Thus, in WebKB-2, a category identifier can be not only a URI or an e-mail address but also the concatenation of the knowledge provider's identifier and a key name, e.g. `wn#dog` and `pm#IR_system` ("wn" refers to WordNet 1.7 and "pm" is the login name of the user represented by the category `philippe.martin@gu.edu.au`). In this third case, the category may still be referenced from outside the KB by prefixing the identifier with the URL of the KB, e.g. `http://www.webkb.org/kb/wn#dog`.

This identifier encoding is used for all the input/output notations in WebKB-2 (FCG, FE, KIF, CGIF, CGLF) except for RDF/XML where URIs have to be used.

In addition to an *identifier*, a category may have various *names* (which may be names of other categories). In FE and FCG, a category identifier may show all the names given by its creator, e.g. `wn#dog__domestic_dog__Canis_familiaris` (at least two underscores must be used to separate the names).

WebKB-2 proposes a special notation to declare categories and links (i.e. second-order relations) between them: the "For Ontology" (FO)[6] notation. It is an extension of the special notation used in CGLF for specialization links between categories. Hence, in the following example, we use FO instead of FE, FCG and CGLF.

For the KIF representation, we chose to use relation types from RDF, RDFS and DAML+OIL rather than from the Frame-ontology and OKBC-ontology of the Ontolingua library, in order to ease the comparison with RDF/XML representations.

For CGIF, we used special relation types (see identifiers in uppercase) and hence *extended* the grammar because this is more in the spirit of the notation (it is supposed to be of higher-level than KIF or RDF/XML and hence already incorporates many special categories such as EQ, GT and LT; such special cases also ease semantic checking and inferencing). We used the same syntactic sugar as in FO to delimit subtype partitions. More details on the rationales and the grammar of our extensions to CGIF can be found on the WebKB site (http://www.webkb.org/doc/CGIF.html).

```
FO:   pm#thing__top_concept_type (^thing that is not a relation^) 29/11/1999
        _  chose (oc fr),   ^  rdfs#class,   !  pm#relation,   =  sowa#T,
        >  {(pm#situation pm#entity)} pm#thing_playing_some_role;

CGIF: [TYPE: pm#thing *x ;thing that is not a relation;]
        (CREATOR ?x philippe.martin@gu.edu.au) (CREATION_DATE ?x 29/11/1999)
        (NAME ?x "thing")  (NAME ?x "top_concept_type")
        (NAME_BY_IN ?x "chose" Olivier.Corby@sophia.inria.fr wn#french)
        (KIND ?x rdfs#class)   (EXCL ?x pm#relation)   (EQ ?x sowa#T)
        (GT ?x {pm#situation pm#entity})  (GT ?x pm#thing_playing_some_role)
```

[6] http://www.webkb.org/doc/F_languages.html#FO

```
KIF:  (defrelation pm#thing ()) (rdfs#class pm#thing)
        (pm#name pm#thing "thing") (pm#name pm#thing "top_concept_type")
        (pm#nameWithCreatorAndLanguage pm#thing "chose"
                               Olivier.Corby@sophia.inria.fr wn#french)
        (dc#Creator pm#thing philippe.martin@gu.edu.au)
        (dc#Date pm#thing "29/11/1999")
        (rdfs#comment pm#thing "thing that is not a relation")
        (daml#disjointWith pm#thing pm#relation)     (= pm#thing sowa#T)
        (daml#disjointUnionOf pm#thing '(pm#situation pm#entity))
        (rdfs#subClassOf pm#thing_playing_some_role pm#thing)
```

In FO, the creator of a link is left implicit when it is also the creator of the category source of the link. Otherwise, the creator has to be specified (as illustrated above for the name "chose"). To represent link creators in the other notations, either contexts or relations with arity higher than two must be used (as illustrated).

"SubtypeOf" links are special cases of *definition of necessary conditions for* (being an instance of) the source categories. Here is an example of how more general cases for the definition of necessary conditions can be represented.

```
E:    A man (according to "pm") has necessarily for father a man.
FCG:  [type pm#man (*x) :=> [*x, pm#father: a pm#man] ]
CGLF: [TYPE: pm#man]->(LT)->[(lambda(*x) [?x]->(pm#father)->[pm#man])]
CGIF: (LT pm#man (lambda (T *x) (pm#father ?x [pm#man])))
KIF:  (defrelation pm#man(?p) :=> (exists((?p2 pm#man)) (pm#father ?p ?p2)))
```

To define sufficient conditions, `GT` and `:<=` may be used instead of `LT` and `:=>`. To define necessary and sufficient conditions, `EQ` and `:=` may be used.

The CG standard is quite incoherent and restrictive about lambda-abstractions and type definitions. The above proposal (with `GT`, `LT`, `EQ`) is the closest generalization we found. We took into account the possible need to contextualize the definitions themselves: with the usual CGLF syntax for type definition with necessary and sufficient conditions (as in: `type pm#red_car is [pm#car]->(pm#chrc)->[pm#red]`), contextualization cannot be done (unless the grammar is extended to accept such definitions as embedded graphs).

The CG standard does not specify how to *define* functional relations (actors), just how to use them. The next example is adapted from [9]: we preferred to use the `IF` construct rather than Sowa's ternary relation `<` and quadrary relation `cond`.

```
E:    The length of a list is 0 if the list is empty,
      otherwise, 1 + the length of the list without its first element
FCG:  [function length (list *l) :-> natural *r
        := [if [l = nil] then [*r = 0] else [*r = 1 + length(rest(*l))] ]]
CGLF: [function length (list *l, natural *n)
       [IF: [?l]->(EQ)->[list:nil] [THEN: [?n]->(EQ)->[number:0] ]
        [ELSE: [?l]-><rest>->[list]-><length>->[natural]-><plus1>->[?n] ] ]
CGIF: [function length (list *l, natural *n)
       [IF: (EQ ?l nil) [THEN: (EQ ?n [number:0])]
        [ELSE: (rest ?l [list:*l2])(length ?l2 [natural:?n2])(plus1 ?n2 ?n)]
       ]]
KIF:  (deffunction length (?l)
        := (if (= ?l nil) 0  (if (list ?l) (1+ (length (rest ?l))))))
```

KIF also has built-in operators (`listOf`, `setOf`) to assemble/decompose lists and sets; e.g.: `(deffunction first (?):= (if (= (listof ?x @items) ?l) ?x)`. CGLF and CGIF need to be extended with such operators.

12 Conclusion

We have shown how FE, FCG, CGLF and KIF can be used in various knowledge representation cases, highlighted some problems of CGLF and CGIF for knowledge representation, exploitation and exchange, and proposed intuitive notations (FE, FCG and FO) covering at least all the presented cases. Although these high-level notations are unlikely to be widely adopted, they show some ways to improve CGIF, CGLF or other notations in readability, expressiveness and "knowledge normalizing effect". They also provide an alternative to graphic notations such as CGDF which suffer from similar problems as CGLF plus the need for specialized tools (graphic notations are not easy to mix and hyperlink with text in documents).

Compared to FE, other controlled English notations are often less formal, e.g. ClearTalk, but closer to English, e.g. Attempto Controlled English [5]. Hence, they are easier to use but permit less (no functions, no categories from different authors or ontologies, etc.) and interpret more. By allowing adjectives, adverbs and verbs, they also do not lead the user to write more explicit and comparable statements [6]. FE and FCG encourage the users to adopt the lexical and ontological conventions that we proposed in [6] to improve knowledge representation and sharing.

We are now working on the import and export of FE, FCG, CGLF, KIF and RDF/XML in WebKB-2, along the lines presented in this article. More information can be found, and testing can be done, at WebKB's site (www.webkb.org).

References

1. The CG specification. http://users.bestweb.net/~sowa/cg/cgstand.htm
2. The KIF specification. http://logic.stanford.edu/kif/dpans.html
   See also: http://www-ksl.stanford.edu/knowledge-sharing/kif/
3. The RDF specification. http://www.w3.org/TR/REC-rdf-syntax/
4. The Knowledge Machine specification. http://www.cs.utexas.edu/users/mfkb/km.html
5. Fuchs, N.E., Schwertel, U., Torge, S.: Controlled Natural Language Can Replace First-Order Logic. In Proc. of ASE'99, 14th IEEE International Conference on Automated Software Engineering, Cocoa Beach, Florida, 1999.
6. Martin, Ph.: Conventions and Notations for Knowledge Representation and Retrieval. In Proc. of ICCS 2000, 8th International Conference on Conceptual Structures, Springer Verlag, LNAI 1867, Darmstadt, Germany (2000) 41–54. http://www.webkb.org/doc/papers/iccs00/
   See also the FE and FCG grammars at http://www.webkb.org/doc/F_languages.html
7. Martin, Ph., Eklund P.: Large-scale cooperatively-built heterogeneous KBs. In Proc. of ICCS 2001, 9th International Conference on Conceptual Structures, Springer Verlag, LNAI 2120, Stanford University, California (2001) 231–244. http://www.webkb.org/doc/papers/iccs01/
8. Sowa, J.F.: Conceptual Graphs Summary. In: Nagle, Nagle, Gerholz, Eklund (eds): Conceptual Structures: Current Research and Practice, Ellis Horwood (1992) 3–51.
9. Sowa, J.F.: Relating Diagrams to Logic. In Proc. of ICCS'93, Springer Verlag, LNAI 699, Laval, Quebec (1993), 1–35.