Enterprise Modeling in Conceptual Graphs Logic.

Ryszard Raban
School of Computing Sciences
University of Technology, Sydney
PO Box 123, Broadway NSW 2007, Australia
richard@socs.uts.edu.au

Abstract: This paper shows how to apply conceptual graphs logic to information systems modeling. The graphs are not used just as another graphical representation of information system requirements but the full power of this graphical logic system has been employed to fully and precisely capture type definitions, referential integrity constraints and global constraints. Such a logic based definition of an information system takes its semantics from a set of facts represented by fully instantiated simple conceptual graphs. An incremental validity checking procedure has been defined and illustrated by an example.

Introduction

Recent efforts in applying conceptual graphs to enterprise modeling concentrated on converting traditional notations into conceptual graphs [Sowa & Zachman92], [Creasy and Ellis93], [Boksenbaum, Carbonneill, Haemmerle & Libourel93], [Gardyn94]. And while this work has proved conclusively that conceptual graphs subsume most of the exiting modeling techniques, one might argue that the full power of the conceptual graphs logic has not been utilized in it.

To model an information system an analyst has to capture its concepts, its structure as well as integrity and global constraints. Conceptual graphs formalism is capable of handling all these aspects of modeling. Equally, it is capable of capturing semantics of the models through the conceptual graphs version of the model theory as described in [Sowa84]. As a result conceptual graphs models are not merely just another representation of information systems requirements but also a set of requirements declarations in logic which can be used to check integrity of data stored as simple conceptual graphs.

Using conceptual graphs in the model theory context creates some terminological problems. A conceptual model is what in the model theory is called a set of formulas making statements about the world [Makowsky92]. On the other hand, a model in the model theory is a set of structures that represent a possible world. The term 'model' means two distinctly different things in the two contexts it is used. In order to avoid any confusion it is assumed for the purpose of this paper that what is called a conceptual model in the information system context and a set of formulas in the model theory context will be called a system definition here. A model as understood in the model theory will be called an information system model. Both information system definition as well as information system model will be represented by conceptual graphs as suggested in [Sowa84].

In order to establish precise understanding of the join and projective extent the first part of this paper defines set referents. Then, using a running example it shows how to create a system definition, evaluate denotations and validate information system facts using the definition.

Referent Notation in Conceptual Graphs

Conceptual graphs are usually drawn in one of two forms, either as diagrams [Sowa84], [Sowa93a] or as linear form expressions [Sowa84], [Wermelinger95]. In the both cases, no complete standard of acceptable forms of referent fields has emerged as yet. A precise and comprehensive method of representing all types of referents is very important in determining semantics of conceptual graphs definitions of information systems. For that reason a variation of the method which was initially suggested in [Sowa93a] has been adopted for the purpose of this paper. This notation uses six basic types of referents:

<u>individual referents:</u>

• **names** which can be used with concepts of type WORD and of all its subtypes. For example, graph
[PERSON]->(NAME)->[WORD: John]
can be used to express that there exists a person called John;

• **measures** which can be used with concepts of type MEASURE, of all its subtypes. For example, graph
[PERSON]->(CHRC)->[HEIGHT]->(MEAS)->[MEASURE: 180 cm]
can be used to say that there exists a person 180 cm tall.
Measures can also be used in the contracted form [Sowa84] with all measurable concepts, like [HEIGHT] in the example, linked directly to a concept of type MEASURE. In this case, however, a measure referent is prefixed by the @ sign to denote that it is a result of contraction. By applying contraction to the previous example the measure referent is moved to the related measurable type concept giving the following graph:
[PERSON] ->(CHRC)->[HEIGHT: @ 180 cm];

• **individual markers** which can be used with concepts that are not of type WORD or type MEASURE, and that are not measurable concepts either. An individual marker represented by a number prefixed by the # sign means a specific instance of the concept. For example, graph
[PERSON: #1082]->(NAME)->[WORD: John]
can be used to say that there is a specific person called John;

<u>generic referent:</u>

- **generic marker** which can be used with any concept, and is the default if no referent is present in a concept. A generic marker means an unspecified instance of the concept. For example, graph
  [PERSON: ∗]->(NAME)->[WORD: John]
  can be used to express that there exists a person called John;
  A generic marker can be restricted to a type consistent individual referent, that is either a name or measure or individual marker;

universal referent:

- **universal quantifier** which can be used with any concept. A universal quantifier means that all individual concepts of a specific type have properties described by a graph containing the concept. For example, graph
  [PERSON: ∀]->(NAME)->[WORD]
  can be used to say that every person has a name.
  A universal quantifier can be restricted to a generic marker or to a type consistent individual referent, that is either name or measure or individual marker;

null referent:

- **null referent** which can be used with any concept. It is mainly utilized to define optionality when included into a disjunctive set. For example, graph
  [PERSON: ∗]->(NAME)->[WORD: ∗ | NULL]
  can be used to say that there is a person with an optional name.

In addition to these singular referents it is convenient to adopt after [Sowa84] two types of set referents. These, however, can be viewed as composite referents because they are always reducible to equivalent representations with basic referents only. The set referents are:

- **collective set** which is a set of any concept type consistent referents except for universal quantifiers and NULL referents. A concept with a set referent can be represented by a set of individual concepts with all relation linked to the original concept repeated for every individual concept. For example, graph
  [PAIR: ∗]->(PART)->[PERSON: {#1082, ∗}]
  is equivalent to
  ```
  [PAIR: ∗]-
          (PART)->[PERSON: #1082]
          (PART)->[PERSON: ∗] .
  ```
  A concept with a collective referent set can be restricted by replacing any of its generic elements with a type consistent individual referent, that is either a name or measure or individual marker. The concept type cannot be, however, automatically restricted to its subtype. In this case type restriction can only be done after the set referent is expanded into individual concepts. This is necessary since not every individual concept resulting from expansion has to be restricted to the same subtype. The previous example can be restricted to the following manner:
  ```
  [PAIR: ∗]-
          (PART)->[MAN: #1082]
          (PART)->[WOMAN: ∗] .
  ```

- **disjunctive set** which is a list if any concept type consistent referents except for universal quantifiers separated by the | sign. When a graph contains a concept with a disjunctive set for a referent it means that any, but only one, graph obtained by replacing the disjunctive set referent by one of its elements can be used for graph operations. For example, graph
  [PERSON: #3201]->(LIKE)->[PERSON: {#1082 | NULL}]
  is equivalent to a disjunction of two graphs:

¬[¬[[PERSON: #3201]->(LIKE)->[PERSON: #1082]]
¬[[PERSON: #3201]->(LIKE)->[PERSON: NULL]]].

A concept with a disjunctive referent set can be restricted by selecting any subset of its elements and/or replacing some or all of its generic markers with a type consistent individual referent. In this case if elements of the disjunctive set are not set referents type restriction can be done the usual way. Otherwise, the prior expansion rule explained for collective sets must be observed.

The shortcuts defined below can also be used for convenience:

- **counted generic set** which means a set with a specific number of referents. For example,
  [PERSON: {∗}@n]
  is equivalent to
  [PERSON: {∗, ∗, ... , ∗,}]
  where ∗ is repeated n times. Obviously, it is not mathematically correct to have the same element more than once in a set, so the explanation provided above should be treated only as an illustrative interpretation of the counted set notation. To specify a partially defined counted generic set, the notation described in the example below should be used:
  [PERSON: {#123, #320, ∗}@5]
  This graph represents two specific persons and additional three unspecified people. The generic marker in this case stands for the remaining unspecified elements of the set.

- **variable generic set** which means a variable number of elements within a specified range. For example,
  [PERSON: {∗}@1...n]
  is equivalent to
  [PERSON: {∗ | {∗}@2 |{∗}@3 | ... |{∗}@n};

- **generic set** which means a set with any number of elements. For example,
  [PERSON: {∗}]
  is equivalent to
  [PERSON: {∗ | {∗}@2 |{∗}@3 | ...}.
  Note that {∗} does not mean a set of one generic element. To specify a set like that, one of the following notations should be used: ∗ or {∗}@1.

This convention sometimes allows for different representations of the same referent set. For example, [PERSON: ∗] means exactly the same as [PERSON: {∗}@1], similarly [PERSON: {NULL | ∗}] can be expressed as [PERSON: {∗}@0...1].

IS Modeling in Logic

Let an information system be defined by system definition $D = \langle S, T, N, M, R \rangle$ where

- T is a type hierarchy,

- R is a relation hierarchy,

- N is a set of names,

- M is a set of measures, and

- $S = \{g_1, g_2, ..., g_n\}$ is a schema which is a consistent and satisfiable set of simple and compound conceptual graphs built out of concepts and relations described by
  - type labels from T and R,

- referents being either generic markers and/or names and/or measures[1], and
- coreference links.

Let us illustrate this on a simple case study. An information system model will be built for a college which provides a three year course to a small number of students. Due to its exclusive character a number of places available in every year of studies is limited. There can be only up to 50 students in the first year, up to 40 in the second and no more than 35 in the third. In the final year every student must have a supervisor for a final project. A teaching staff member cannot supervise more than five students. For simplicity let's assume that every person registered in the system must have a name and an optional telephone number. A student identification number and a year of study must be stored for every student, and a staff identification number and a date of hire must be registred for every teaching staff member.

The type hierarchy *T* in this case contains the following type labels: ENTITY > PERSON > STUDENT and PERSON > STAFF. The relation hierarchy *R* is made of three relations CHRC, NAME and SUPER. Using these types the system requirements of the case study can be represented by the following set *S* of conceptual graphs:

```
(g1)   {¬[[PERSON: *x]  ¬[[ENTITY: ?x]->(NAME)->[WORD]
                                       (CHRC)->[PHONE: * | NULL]]],
(g2)    ¬[[STUDENT: *x] ¬[[PERSON: ?x]->(CHRC)->[STUD-ID]
                                         (CHRC)->[STUD-YEAR]]],
(g3)    ¬[[STAFF: *x]     ¬[[PERSON: ?x]->(CHRC)->[STAFF-ID]
                                         (CHRC)->[DATE-HIRED]]],
(g4)    ¬[[STUDENT: *x]->(CHRC)->[STUD-YEAR: 3]
                  ¬[[STAFF]->(SUPER)->[STUDENT: ?x]]],
(g5)    ¬[[STAFF: *x]  ¬[[STAFF: ?x]->(SUPER)->[STUDENT: {*}0...5]]],
(g6)      [STUDENT: {*}0...50]->(CHRC)->[STUD-YEAR: 1],
(g7)      [STUDENT: {*}0...40]->(CHRC)->[STUD-YEAR: 2],
(g8)      [STUDENT: {*}0...35]->(CHRC)->[STUD-YEAR: 3],
```

Graphs *g1*, *g2* and *g3* define types PERSON, STUDENT and STAFF: *g1* states that every person is an entity, *g2* says that every student is a person, and *g3* states that every staff is a person. Graphs *g4* and *g5* impose referential integrity constraints on relation (SUPER): *g4* says that every third year student has to have a supervisor and *g5* represents the condition that a supervisor cannot supervise more than five students. And finally, student yearly quotas are represented by graphs *g6*, *g7* and *g8*. In this way, a complete information system model with type definitions, referential integrity and global constrains has been represented by a set of conceptual graphs.

Semantics of System Definition

D represents a logic based definition of an information system. A set of propositions in schema *S* contains statements about the system, its concepts, relations and constraints. This conceptual graphs logic has to be complemented with semantics. For this purpose let us define an information system model $W = \langle F, I \rangle$, where

[1]Universal quatifier is not used explicitly in defining schemas. It can be, however, represented by an oddly enclosed dominant concept in a line of identity. For example,
[PERSON: ∀]->(NAME)->[WORD]
is equivalent to
¬[[PERSON: *x] ¬[[PERSON: ?x]->(NAME)->[WORD]]].

- $F = \{f_1, f_2, ..., f_n\}$ is a set of simple graphs asserting atomic facts. Graphs in F contain fully instantiated concepts and relations defined in D,

- I is a set of individual referents (markers, names and measures) used in F.

Since schema S is satisfiable there has to be a model W such that the denotation of S, δS, is **true** in F. The evaluation game based method of evaluating denotations for conceptual graphs has been given in [Sowa84]. A model W which satisfies a schema S of D is called a model of information system D, or simply a model of D. A set of all models of a given information system D is denoted by $\mathcal{W}_D$.

Any model of D from which no single graph, concept or relation can be removed without making δS **false** in the reduced model is called a minimal model of D. For every definition D there is a set of minimal models. The minimal models usually represent the set up data of an information system, like for example exchange rates, company name or default values. Model

($W0$) $\langle$\{[STUD-YEAR: 1], [STUD-YEAR: 2], [STUD-YEAR: 3]\}, \{\}$\rangle$

is the only minimal model of the case study information system.

During its life time an information system is constantly updated and evolves through a possibly infinite sequence of information system models W_0, W_1, W_2, ... , where W_0 is one of minimal models of D. An information system maintains integrity if and only if for all i, $W_i \in \mathcal{W}_D$. The rest of this paper will demonstrate how the denotation evaluation procedure is used to maintain the integrity of an evolving information system defined by D. The simplest approach in evaluating changes could be to check if a modified model still satisfied the definition. This, however, means evaluating the entire model after every change which will become more and more costly and inefficient as the model grows larger and larger. For that reason this paper adopts another strategy based on checking only increments by which the model is being changed.

In order to establish increments by which an information system model can be changed let us have a look at different kinds of concepts populating the model. Following the ontological criteria of the foundation and semantic rigidity introduced in [Guarino92] it is obvious that some concepts can exist independently while others necessarily depend on existence of other concepts. In a particular universe of discourse the foundation results in relationship between concepts. Some concepts called properties, like color or height, are special as they are dependent but semantically rigid and through their relational interpretations represent knowledge about terms used to describe a system. Such terms can be linked together via structural relations. In the conceptual graphs formalism concepts of names and measurable types are properties and all the other concepts are terms which are called later objects. And since properties convey the knowledge about objects, properties can only be related to objects. On the other hand, objects can also be related to each other representing the relational knowledge. This leads to two types of relations: these linking objects with properties called later type definition relations and those linking objects with other objects called later structural relations.

Properties cannot be added to an information systems model on their own since they are dependent on objects that are described by them. Objects, however, when added into a model must have all required properties, and also all structural relations with objects as required by referential integrity constraints.

Evaluating denotations.

The evaluation game decides whether a set of graphs in $S = \{g_1, g_2, ..., g_n\}$ is **true** or **false** in a set of facts $F = \{f_1, f_2, ..., f_n\}$. The game is played by proposer and skeptic who take turns in a sequence of moves consisting of project, select and reduce operations always performed in this order. The result of the game is not decided by an outcome of a particular play but instead by the fact which side has a winning strategy in the game. If proposer has a winning strategy then the denotation is **true**, if skeptic has one then the denotation is **false**. In presenting the game πg will be used to denote a projection selected from the projective extent of g in T, and $g\#$ will denote a graph obtained by copying referents from dominating concepts into concepts in a graph g. Also, it is assumed that a compound graph takes always the form of $\neg[p_1, p_2, ..., p_n]$ and its referent $\{p_1, p_2, ..., p_n\}$ must consist of at least a single simple graph or a couple of compound graphs. This does not reflect on generality of the representation as $\neg[\neg[p]]$ can always be reduced to just p.

For a set $S = \{g_1,..., g_n\}$ of graphs the game is played as shown in TABLE I and II.

TABLE I. Evaluation of simple graphs.

	PLAYER	MOVE	OUTPUT GRAPH	
START			$\{g_1,..., g_n\}$	
1	Prop	Proj	$\{g_1,...,\pi g_i, ..., g_j\#, ..., g_n\}$	
2	**Skept**	**Sel**	$\{g_h\}$ - g_h is a simple graph	$\{\pi g_i\}$ - g_i is a simple graph
3	Prop	Red	{} **Skept wins**	$\{\pi g_i\}$ **Prop wins**

Since skeptic has the first selection it initially controls the game. Any graphs in S with the denotations **false** wins the game for skeptic. To block this possibility all graphs in S must be **true** in F. To ensure that, firstly (as shown in TABLE I) all simple graphs must have a non-empty projective extent in F. For that reason the minimal model of the case study information system had to have the three initial elements to make sure that graphs g6, g7 and g8 could be successfully projected into it. And secondly, for all compound graphs proposer must have a winning strategy. Since proposer is now in control (move 5 in TABLE II) it can win either by selecting a simple graph with an empty projective extent or, if all of them have non-empty projective extents, select one of the compound graphs for which the denotation of its referent is **true**. From this point the evaluation of a set of graphs being the referent of the compound graph selected goes through exactly the same steps as for schema S.

TABLE II. Evaluation of compound graphs.

	PLAYER	MOVE	OUTPUT GRAPH	
START			$\{g_1, ..., g_n\}$	
1	Prop	Proj	$\{g_1,...,g_i, ..., g_j\#, ..., g_n\}$	
2	**Skept**	**Sel**	$\{g_p\}$ - g_p is a compound graph $\neg[p_1,..., p_m]$	
3	Prop	Red	$\{p_1,..., p_m\}$	
4	Skept	Proj	$\{p_1,..., p_s\#, ..., \pi p_r, ,...,p_m\}$	
5a	**Prop**	**Sel**	$\{p_k\}$ - p_k is a simple graph	$\{\pi p_r\}$ - p_r, is a simple graph
6a	Skept	Red	{} **Prop wins**	$\{\pi p_r\}$ **Skept wins**
5b	**Prop**	**Sel**	$\{p_s\}$- p_s is a compound graph $\neg[q_1, ..., q_l]$	
6b	Skept	Red	$\{q_1, ..., q_l\}$	
7b	Prop	Proj	$\{q_1, ..., q_l\}$ **which is the starting problem**	

Under the assumption that the propagation of referents in project steps of the evaluation game is always performed properly, the following two observations can be made:

- a set of graphs being the referent of an evenly enclosed negative context has the denotation **true** if all of the graphs have their denotations **true**, and
- a set of graphs being the referent of an oddly enclosed negative context has the denotation **true** if at least one of the graphs has the denotation false, that is either one of simple graphs in the set has an empty projective extent, or the referent of one of compound graphs in the set has the denotation **true**.

These observations can be useful in establishing the most efficient strategy for proposer and skeptic who, when in control, try to find a graph with the denotations **false**. The best course of action for them is to start checking simple graphs first and then progressively go from less to more complex graphs.

Enforcing Typing

All type defining graphs are in the form of $\neg[\, p\ \neg[q]]$ where q contains only type definition relations between an object (genus) and its properties (differencia). An object in graph p is of the type being defined. Graph *g1* defines a person as an entity with a name and a phone as properties. Graph *g2* uses the type defined in *g2* to define a student as a person with additional properties: student identification number and year of study.

The addition of a new object must create a new information system model which still satisfies the information system definition. To ensure that all type defining implications must be **true**. This, in turn, requires that in all type defining compound graphs in the form of $\neg[\, p\ \neg[q]]$ must have either p **false** or both p and q **true**. In order to illustrate the type checking using system definition *D*, let us consider an addition of a new student

```
(f1)    [STUDENT: #123]-
                (NAME)->[WORD: John]
                (CHRC)->[STUD-YEAR: 3].
```

to the following model

```
(W1)    {{[STUD-YEAR: 1], [STUD-YEAR: 2], [STUD-YEAR: 3]
          [STAFF: #432]-
                        (NAME)->[WORD: Mary]
                        (CHRC)->[PHONE: 663-6754]
                        (CHRC)->[STAFF-ID: 723]
                        (CHRC)->[DATE-HIRED: 95/10{/20]
          [STAFF: #521]-
                        (NAME)->[WORD: George]
                        (CHRC)->[STAFF-ID: 320]
                        (CHRC)->[DATE-HIRED: 92/02/12]}, {#432, #521}}
```

of the college system. Model *W1* satisfies the system definition and the additions of the new fact must lead to a model that also satisties the definition.

The premise of graph *g3* has an empty projection in *f1*, so graph *g3* is satisfied in *f1*. The premises of graphs *g1* and *g2* are satisfied in *f1* and therefore their conclusions must be satisfied in *f1* if the graphs are to be satisfied in *f1*. In order to perform the checking let us first restrict the premises of *g1* and *g2* to their respective projections in *f1*, propagate the referent to the inner negative contexts and perform join of the resulting conclusion graphs. This leads to the following graph:

```
(g9)    [PERSON: #123]-
                (NAME)->[WORD]
                (CHRC)->[PHONE: * | NULL]
                (CHRC)->[STUD-ID]
                (CHRC)->[STUD-YEAR].
```

By using the join not only properties of the type being checked but also all properties inherited from its supertypes are included into the checking process. Obviously, if the joined graph is satisfied in a new fact, so are all the graphs that have been used to create the join. In order to check whether a new fact graph conforms with the combined definition requirements the fact graph has to be maximally joined with the joined definition graph. The maximal join *g10* and *f1* produces the following graph:

```
(g10)   {[STUDENT: #123]-
                (NAME)->[WORD: John]
                (CHRC)->[STUD-ID: *]
                (CHRC)->[STUD-YEAR: 3]}.
```

A new fact graph is valid if all its concepts are covered by the maximal join with a combined definition graph, and all concepts of a combined definition graph are also covered by the join. A concept *c* in a graph *u* is covered by a maximal join *w* of graph *u* and graph *v* if there is a concept *c'* in *v* such that *c* and *c'* were joined together into a concept in *w*. Otherwise a concept is uncovered. Concepts of a fact graph not covered by the maximal join represent some spurious elements in the fact graph not present in the type definitions. On the other hand, all concepts of a combined definition graph not covered by the maximal join are those elements required by the type definitions that are not present in the fact graph. Graph *g12* contains one uncovered concept, STUD-ID, which appears with a generic referent, indicating that fact *f1* does not contain one required property. And therefore fact graph *f1* is invalid. In order to rectify the problem fact graph *f1* has to be corrected into the following graph:

```
(f2)    [STUDENT: #123]-
                (NAME)->[WORD: John]
                (CHRC)->[STUD-ID:123]
                (CHRC)->[STUD-YEAR: 3].
```

It is left to the reader to follow the previous steps and check validity of this new fact. The validity checking and correction process for a fact graph *f* can be summarized by the following procedure:

1° select all type defining graphs with premises having non empty projections in fact *f*,

2° propagate individual markers,

3° create graph *u* by joining all the conclusions of the selected graphs,

4° create graph *w* by performing a maximal join of *u* and *f*, 5° if all concepts in *u* are covered by *w*, and all concepts in *f* are covered by *w* then graph *f* is valid and the checking procedure ends, otherwise

6° remove from graph *f* all concepts not covered by *w*, if any,

7° attach to graph *f* all concepts in *u* which are not covered by *w*, if any,

8° go to step 4°.

Referential Integrity Checking

Type enforcing is concerned with type definition relations that can be checked in the context of a single object. When integrity constraints are checked, structural relations between objects are involved. In this case, however, new structured relations might be dependent on existing ones, and therefore the checking must include relevant elements of graphs from the fact set of *F*.

There are two graphs *g4* and *g5* that represent referential integrity constraints. the both of them have to be **true** if the new fact is to comply with the constraints. This time no graph can be initially discarded as irrelevant since the context in which the new object appears can change as the checking progresses. In the first stage the premise of graph *g4* has non-empty projection in fact graph *f2* and therefore has to be considered as relevant for integrity checking. For graph *g4* to be **true** its conclusion in the nested negative context has to be **true**. In order to establish what relations are required to make the conclusion of graph *g5* **true,** first a maximal join between fact graph *f2* and the conclusion graph has to be performed leading to the following result:

```
(g11)   [STUDENT: #123]-
               (NAME)->[WORD: John]
               (CHRC)->[STUD-ID:123]
               (CHRC)->[STUD-YEAR: 3]
               (SUPER)<-[STAFF: *].
```

Graph *g11* contains one generic object related to the new student concept. This is a result of concept [STAFF] in the conclusion of graph *g4* which is not covered by the maximal join *g11*. It indicates that to maintain referential integrity a relation between student and staff has to be establish. The new relation can be created with an already registered staff member or with a new one. In order to establish what existing staff members can be considered for the relation, the next step in the integrity rule checking procedure is to perform projective extent of graph *g11* into the facts in model *W1* and joint the result with graph *g11*. This yields the following graph:

```
(g11)   [STUDENT: #123]-
               (NAME)->[WORD: John]
               (CHRC)->[STUD-ID:123]
               (CHRC)->[STUD-YEAR: 3]
               (SUPER)<-[STAFF: {#432 | #521 | new}].
```

Note that different elements in the projective extent set appear here in concept [STAFF] as a disjunctive set referent supplemented by the *new* option. This indicates to the user of the system that a relation with a supervising staff has to be establish and that already recorded staff members can be used for this purpose or a new one created. Let us assume that the decision has been made to select the first option from the referent list. Now, the fact graph takes the following form:

```
(f3)    [STUDENT: #123]-
              (NAME)->[WORD: John]
              (CHRC)->[STUD-ID:123]
              (CHRC)->[STUD-YEAR: 3]
              (SUPER)<-[STAFF: #432].
```

In this way the referential integrity rule expressed in graph *g4* is satisfied. But at the same time the rule represented by graph *g5* becomes relevant since its premise has now a non-empty projection in fact graph *f3*. So after propagation of the individual marker, maximal join of the premise of graph *g5* and fact graph *f3* is performed giving the following graph:

```
(g12)   [STAFF: #432]->(SUPER)->[STUDENT: {#123, *}@1...5]
```

The maximal join instantiated one of the possible five instances of concept [STUDENT] that has been indicated by inserting the individual marker into the set referent. Also the range of the set referent has been narrowed as already one individual marker rules out the zero option. Joining graph g12 with graphs of a projective extent of graph *g12* in the facts in model *W1* does not change graph *g12* indicating that the allowed number of supervised students is not exceeded. If, however, a number of supervised student was greater than the allowed limit then a contradiction would be raised and the process would have to be backtracked to the supervisor selection point and another supervisor would have to be selected.

The above process of referential integrity checking of a fact graph *f* can be summarized by the following procedure:

1° from the referential integrity constraints graphs select a graph with a non-empty projection of its premise in *f*,

2° propagate individual markers creating conclusion graph *u*,

3° create a maximal join *w* of *f* and *u*; represent all joined alternatives of any disjunctive set referent as a new disjunctive set referent and instantiate respective elements of set referents,

4° if contradiction occurs backtrack to the point where the contradicting concept or relation has been instantiated, correct and proceed with checking,

5° perform a maximal join of *w* with graphs in projective extent $\Pi(w, F)$; show all alternative outcomes as disjunctive set referents supplemented by the *new* option,

6° if contradiction occurs backtrack to the point where the contradicting concept or relation has been instantiated, correct and proceed with checking,

7° prompt the user to make decision concerning the selection from the alternate options and if necessary to create new objects that in effect modify fact graph *f*,

8° select a next unused integrity constraint graph with non-empty projection of its premise in graph *f* and go to step 2°,

9° if no more integrity constraint graphs with non-empty projection the process is finished and the new fact graph is valid.

Global Constraints Checking

The global constraints are represented in this case by a set of simple graphs *g6*, *g7* and *g8*. All of them must be **true** if the entire definition is to be satisfied in a new model resulting from the addition of fact graph *f3*. In this case only the graphs that have non-empty projections in fact graph *f3* need to be considered. The truth value of the others will not be affected by the new graph and they must have been **true** for model *W1* which is now to be extended with fact graph *f3*.

Therefore, the only global constraints graph to be checked is *g8*. To establish if it is still going to be **true** after the new fact is added to the model maximal join of graph *g8* and fact graph *f3* has to be performed producing the following result:

(*g13*) [STUDENT: {#123, *}@1...35]->(CHRC)->[STUD-YEAR: 3].

And now, in the similar way to the referential integrity checking procedure, graph *g13* must be joined with all graphs of a projective extent of graph *g13* in the set of facts in model *W1*. This does not lead to any changes to graph *g13* and does not create any contradictions. So, fact graph *f3* is valid and can be added to model *W1* giving a new modified model as follows:

```
(W1)   {{[STUD-YEAR: 1], [STUD-YEAR: 2], [STUD-YEAR: 3]
         [STAFF: #432]-
                        (NAME)->[WORD: Mary]
                        (CHRC)->[PHONE: 663-6754]
                        (CHRC)->[STAFF-ID: 723]
                        (CHRC)->[DATE-HIRED: 95/10{/20]
          [STAFF: #521]-
                        (NAME)->[WORD: George]
                        (CHRC)->[STAFF-ID: 320]
                        (CHRC)->[DATE-HIRED: 92/02/12]}
         [STUDENT: #123]-
                        (NAME)->[WORD: John]
                        (CHRC)->[STUD-ID:123]
                        (CHRC)->[STUD-YEAR: 3]
                        (SUPER)<-[STAFF: #432], {#123, #432, #521}}
```

The above process of global constraints checking of a fact graph *f* can be summarized by the following procedure:

1° from the global constraints graphs select a graph *u* with a non empty projection in fact graph *f*,

2° create a maximal join *w* of *f* and *u*; represent all joined alternatives of any disjunctive set referent as a new disjunctive set referent and instantiate respective elements of set referents,

3° if contradiction occurs backtrack to the point where the contradicting concept or relation has been instantiated, correct and proceed with checking,

4° perform a maximal join of *w* with graphs in projective extent $\Pi(w, F)$; show all alternative outcomes as disjunctive set referents supplemented by the *new* option,

5° if contradiction occurs backtrack to the point where the contradicting concept or relation has been instantiated, correct and proceed with checking, 7° prompt the user to make decision concerning the selection from the alternate options and if necessary create new objects, that in effect modify fact graph *f*,

8° select a next unused global constraints graph with non-empty projection in graph *f* and go to step 2°,

9° if no more global constraint graphs with non empty projection the process is finished.

Conclusion

This paper shows how conceptual graphs logic can be applied to information systems modeling. Conceptual graphs logic has been used to create information system definition and simple, fully instantiated conceptual graphs have been used to represent fact stored in the system. Incremental methods of checking typing, referential integrity constraints and global constraints have been introduced and illustrated by an example.

At this stage the information system definition has been limited to simple graphs and compound graph in the form $\neg[\ p\ \neg[q]]$ which while sufficient in defining types might prove too restrictive for representing referential integrity constraints and global constraints. Of course more complex constraints can always be checked by the evaluation game introduced by [Sowa84] and discussed in detail in the paper. This however might not be practical for checking incremental changes to an information system model.

The paper addressed checking data to be added into an information system model. This leaves the problems of deleting facts from and updating facts in the model. These issues have not been covered here.

References

[Creasy & Ellis93] Creasy, P., and Ellis, G. A Conceptual Graphs Approach to Conceptual Schema Integration. In Mineau, G.W., Moulin, B., and Sowa, J.F., editors, *Conceptual Graphs for Knowledge Representation*, pages 126-141. Springer-Verlag, Berlin, 1993.

[Boksenbaum, Carbonneill, Haemmerle & Libourel93] Boksenbaum, C., Carbonneill, B., Haemmerle, O., and Libourel, T. Conceptual Graphs for Relational Databases. In Mineau, G.W., Moulin, B., and Sowa, J.F., editors, *Conceptual Graphs for Knowledge Representation*, pages 345-360. Springer-Verlag, Berlin, 1993.

[Esch & Levinson95] Esch, J. and Levinson, R. An Implementation Model for Contexts and Negation in Conceptual Graphs. In Ellis, G., Levinson, R., Rich W., and Sowa, J.F.., editors, *Conceptual Structures: Applications, Implementation and Theory* , pages 247-262. Springer-Verlag, Berlin, 1995.

[Esch & Levinson96] Esch, J. and Levinson, R. Propagating Truth and Detecting Contradiction in Conceptual Graph Databases. In Eklund, P.W., Ellis, G., and Mann, G., editors, *Conceptual Structures: Knowledge Representation as Interlingua* , pages 229-247. Springer-Verlag, Berlin, 1996.

[Gardyn94] Gardyn, E. Multiple Information Models. In Ellis, G., and Eklund, P., editors, *Proceedings of the 1st Australian Conceptual Structures Workshop*, pages 148-164. University of New England, Armidale, Australia, 1994.

[Rumbaugh91] Rumbaugh, J. et al. *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs. NJ, 1991.

[Makowsky92] Makowsky, J.A. Model Theory and Computer Science: An Appetizer. In Abramsky, S., Gabbay, D.M., and Maibaum, T.S.E., editors, *Handbook of Logic in Computer Science*. Vol 1, Clarendon Press, Oxford, 1992.

[Sowa84] Sowa, J.F. *Conceptual Structures - Information Processing in Mind and Machine*. Addison-Wesley, Reading, Massachusetts, 1984.

[Sowa93a] Sowa, J.F. Relating Diagrams to Logic. In Mineau, G.W., Moulin, B., and Sowa, J.F., editors, *Conceptual Graphs for Knowledge Representation*, pages 345-360. Springer-Verlag, Berlin, 1993.

[Sowa93b] Sowa, J.F. Logical Foundations for Representing object-Oriented Systems. *Journal of Experimental and Theoretical Artificial Intelligence*, **5**(1993), pages 237-261.

[Sowa & Zachman92] Sowa, J.F., and Zachman, J.A. Extending and Formalizing the framework for Information Systems Architecture. *IBM Systems Journal* **31**, 3.

[Wermelinger95] Wermelinger, M. Conceptual Structures Linear Notation - Proposal for Pierce. In Eklung, P., and Ellis, G., editors, *Proceedings of the Fourth International Workshop on PIERCE: A Conceptual Graphs Workbench*, pages 106-125. University of California Santa Cruz, CA, August, 1995.