Difference Graphs

Harry S. Delugach and Aldo de Moor

Department of Computer Science
N300 Technology Hall
University of Alabama in Huntsville
Huntsville, AL 35899 USA
delugach@ cs.uah.edu

STAR Lab
Vrije Universiteit Brussel
Pleinlaan 2, Gebouw G-10
1050 Brussels, Belgium
ademoor@vub.ac.be

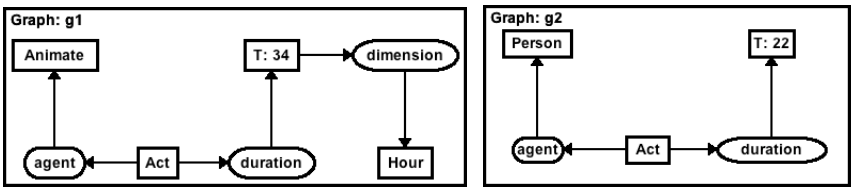
Abstract. In the conceptual structures community, much effort and attention has been focused on determining areas of commonality between two or more conceptual graphs. Less attention has been paid to another aspect of comparison, namely, what is *different* between two or more graphs. This paper explores a technique called difference graphs for determining and representing the differences between two graphs, both visually and formally. Difference graphs can be used in comparative analyses between mental models, process vs. practice models, and any domain in which multiple conceptual graphs can be obtained.

1 Introduction

In the conceptual structures community, much effort and attention has been focused on determining areas of commonality between two or more conceptual graphs; e.g., matching, similarity, overlap, etc. Less attention has been paid to another aspect of comparison, namely, what is explicitly *different* between two or more graphs. Intuitively, what is different comprises whatever *does not* match, whatever *is not* similar or whatever *does not* overlap.

There are many applications where it would be useful to understand the difference between two graphs. We could compare an observation with a model and identify where they do not match or compare two models or compare two observations and see how a situation has changed over time. Team-based large-scale development could benefit from a comparison of different developers mental models. Instructors could compare their own "right" answers with those submitted by students.

For instance, Figure 1(a) and (b) show two graphs that are different. How might the differences be characterized?



(a) (b)

Figure 1. Two different graphs.

A key feature of our comparisons is that the comparison process does not imply any "rightness" or "wrongness" with respect to one graph or the other. For example, if an observation does not match our model, it's possible that at least one of them is wrong, but merely comparing them does not tell us which one(s). The comparison processes we describe in this paper are for the purposes of determining differences between graphs, not deciding which graph to believe.

Another key feature of our comparison method is that we deliberately relinquish the claim of having a unique and deterministic solution, so that we speak of "a" difference, not "the" difference. This is to remind readers that (for any given pair of graphs) many distinct (yet correct) differences can be identified.

There are two main issues identified in this process:

- How to compute a difference? Most operations on graphs are aimed at establishing the common elements; e.g., finding the least common generalization or most common specialization. Since we are interested in the difference for human understanding and interpretation, we are satisfied (at present) with using any correct difference for our purposes.
- How to display (or otherwise convey) a difference? Semantically a difference graph (see below) may have the same constituents as its original graph but a visual editor allows coloring or other syntactic adornments to easily convey the difference.

The organization of the paper is first to explain how graph differences are computed with difference sequences; this is one technical contribution of the paper. We then explain how to display the differences through a coloring process; this is the second contribution of the paper. We then apply our technique to a real-world analysis problem involving model comparison, and offer some conclusions.

2 Computing Graph Differences

In order to compute graph differences, we first make an important pragmatic assumption:

Assumption. There need not be a provably unique difference characterization between any two graphs. Since our purpose is for humans to analyze the differences, we can allow that, for a given pair of graphs, more than one such difference is possible. It may be useful in the future to establish a formal mathematical description of what it means to be an "optimal" difference description, but for now we characterize the differences between two graphs by describing any sequence of changes needed to transform one into the other. In this way, we avoid having to justify that our approach is tractable in all cases; we need only show that it leads to a correct difference description.

Definition. A *difference operation* is any one of the transformation operations shown in Figure 2. We call the starting graph the *source* graph and the resulting graph the *result* graph. These operations transform part of one graph into part of another graph.

Definition. The *inverse* of a difference operation is an operation which will transform the result of a difference operation back into its original graph. Note that in Figure 2, each operation has an inverse operation associated with it; later, we will show how the inverse operations are used.

Difference operation	Name	Description	Inverse
Specialize concept	**S-C**	Replace with subtype, or Add new referent	**G-C**
Specialize relation	**S-R**	Replace with subtype	**G-R**
Generalize concept	**G-C**	Replace with supertype, or Delete old referent	**S-C**
Generalize relation	**G-R**	Replace with supertype	**S-R**
Insert subgraph	**I-G**	Add subgraph components	**D-G**
Delete subgraph	**D-G**	Remove subgraph components	**I-G**
Insert co-referent link	**I-L**	Link two or more concepts	**D-L**
Delete co-referent link	**D-L**	Remove link from two or more concepts	**I-L**
Move into context	**M-I**	Move a subgraph components into a context	**M-O**
Move from context	**M-O**	Move a subgraph from a context	**M-I**

Figure 2. Difference operations and their inverses.

Note that we make no claim about the truth-preserving characteristics of these steps: this clearly distinguishes them from the usual notion of a conceptual graph "transformation rule" which is intended to preserve the original graph's meaning, as in work such as [1]. Here we have explicit change steps, which we expect *will change* the intent of the original graph – that is why we are doing it.

Each difference operation represents a small change. The overall difference between two graphs is thereby captured by the accumulation of many small differences. We offer some brief definitions.

Definition. A *difference sequence DS(G, H)* between graphs *G* and *H* is a sequence of zero or more difference operations applied in order to *G* that will transform it into *H*.

Definition. The *inverse difference sequence* (or simply, *inverse*) of a difference sequence *DS(G, H)* is a sequence of zero or more difference operations applied in order to *H* that will transform it back into *G*. This sequence is shown as $DS^{-1}(G, H)$.

For any difference sequence DS consisting of steps s_1, s_2, ..., s_n, there exists an inverse sequence DS^{-1} consisting of steps s_n^{-1}, s_n^{-1}, … s_2^{-1}, s_1^{-1}, where each s_i^{-1} represents the inverse of step s_i. We can thus completely specify the difference between two graphs by specifying either *DS* or DS^{-1}; we sometimes use both for convenience.

Notation. Two or more difference sequences can be concatenated to form a single sequence. We show concatenation between sequences with a "+" symbol, so that *DS(G, H) + DS(H, J) = DS(G, J)*.

Notation. A least common generalization between graphs *G* and *H* is denoted by *LCG(G, H)*. The least common generalization notion is the same as in the standard conceptual graph literature e.g., [2], [3].

There are several different ways to obtain a difference sequence. We partition the procedure by assuming an intermediate graph – namely an LCG. This provides a convenient starting point, where we begin by obtaining an LCG, then we generalize "up" to that LCG from the first graph, and then specialize "down" to the second graph[1], as shown in Figure 3. We use the LCG because it is a well-understood property known to be computable for any two graphs [4] [5] and therefore we do not need to justify the choice of any particular LCG. A consequence of this decision is that there may be more than one valid difference sequence for a given pair of graphs.[2]

The algorithm is organized into four steps:

- Obtain the LCG of the two graphs
- Find a sequence of difference operations that transforms the first graph into the LCG
- Find a sequence of difference operations that transforms the LCG into the second graph.
- Concatenate the two sequences.

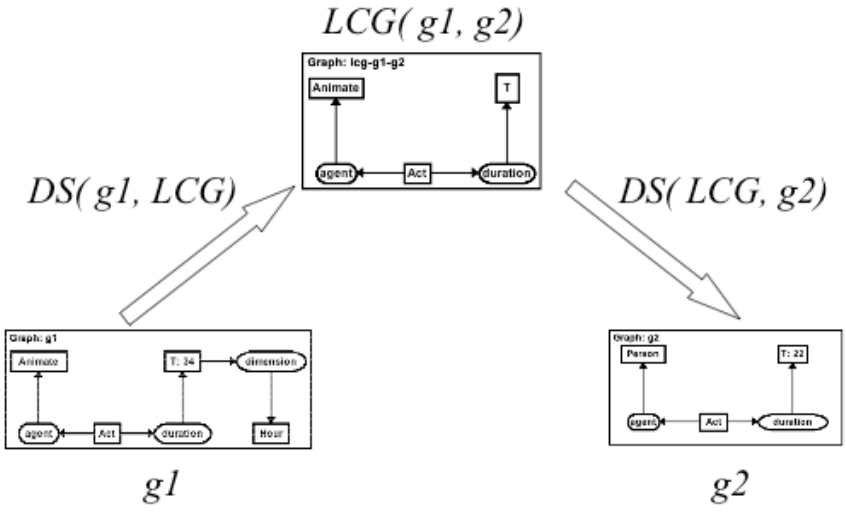


Figure 3. General form of the algorithm.

[1] We could just as easily start with the second graph, generalize to an LCG, and then specialize to the first graph. The choice is arbitrary. The important point is that for any given pair of graphs, there exists at least one LCG to serve as a "bridge".

[2] Some readers may not be comfortable that there is more than one LCG for a given pair of graphs and therefore there may be more than one correct difference sequence. Because we use the difference graphs for human analysis, we can allow for some non-determinism in obtaining the graphs.

An example of applying the algorithm is shown in the rest of this section. We obtain a set of operations to transform the first graph into one of its LCG's, then obtain a set of operations to transform that same LCG into the second graph. The concatenation of these two sequences forms an overall difference sequence.

2.1 Obtain an LCG of the two graphs

A least common generalization (*LCG*) of the graphs in Figure 1(a) and (b) is shown in Figure 4.

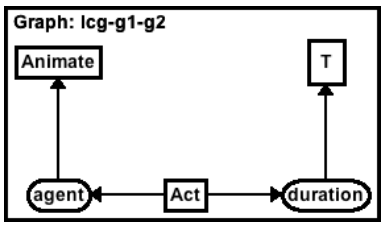


Figure 4. Least Common Generalization (LCG) of two graphs.

2.2 Transform first graph into LCG

Since we begin this part of the algorithm with the *LCG*, we will determine *DS(LCG, g1)* and then get its *inverse* DS^{-1}*(LCG, g1) = DS(g1, LCG)*. This is, we will first obtain a sequence of operations that transforms the *LCG* into *g1*. This is more convenient because we intend to find the difference sequence from the LCG to the second graph using the same process. We then invert the sequence (i.e., invert each of its operations and reverse the order). To transform the *LCG* graph into each of the two graphs, we list the following steps (each of which specializes the *LCG* graph):

DS(LCG, g1) :
S-C: Specialize **T** to **T: 34**
I-G: Insert **(dimension) -> [Hour]**

Inverting this sequence, we get DS^{-1}*(LCG, g1) = DS(g1, LCG)*.

DS(g1, LCG) :
G-C: Generalize **T:34** to **T**
D-G: Delete **(dimension) -> [Hour]**

2.3 Transform the LCG into the second graph

The second sub-sequence is somewhat easier than the first, only because we do not have to invert it as for *DS(g1, LCG)*.

DS(LCG, g2) :
S-C: Specialize **Animate** to **Person**
S-C: Specialize **T** to **T: 22**

2.4 Concatenate the two sub-sequences

Once the two sub-sequences are obtained, their concatenation is straightforward:

$$DS(g1, g2) = DS(g1, LCG) + DS(LCG, g2)$$

A complete difference sequence *DS(g1, g2)* is therefore:

DS(g1, g2) :
G-C: Generalize **T:34** to **T**
D-G: Delete **(dimension) -> [Hour]**
S-C: Specialize **Animate** to **Person**
S-C: Specialize **T** to **T:22**

This section demonstrated a straightforward method of obtaining a difference sequence between conceptual graphs that can be used to describe in a formal way how two graphs are different. While these operations are fairly simple, their combinations can be quite powerful in capturing graph differences. In particular, we intend to apply these differences to support humans in analyzing pairs of graphs (see sec. 4 below). The next section describes how we display the differences for human use and benefit.

3 Displaying Differences For Human Interpretation

While our calculation of a difference sequence may be logically adequate, we are interested in using graph differences to guide human users in analyzing the graphs. We therefore want to highlight the different parts so that human users can quickly determine what has changed between the graphs. We adopt the convention of *coloring*, i.e., using graphical shading in the displayed graphs. We show difference graphs in Figure 5 and Figure 10. We use gray for the un-changed parts of the graphs and white for the changed parts, but other highlighting strategies could be used.

Displaying the differences between two graphs can be done from the perspective of either of the two graphs. This means that two distinct difference graphs are meaningful for any pair of graphs.

Definition. A *difference graph* between two graphs *g1* and *g2* is a copy of either *g1* or *g2* with the common elements of the graphs shaded in one color, and the different elements shaded in another color. For example, given the two graphs *g1* and *g2* in Figure 5(a) and (b), two difference graphs are shown in Figure 5(c) and (d).

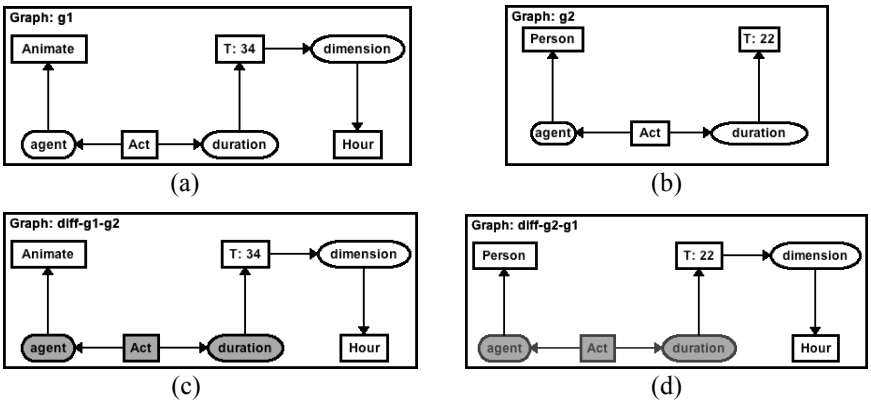


Figure 5. Displaying the differences between graphs.

We adopt the naming convention that a difference graph is named "diff" followed by the name of the original graph to be displayed, followed by the name of the second graph whose differences are to be highlighted. Therefore **diff-g2-g1** is a copy of *g2*, with the differences from *g1* highlighted.

One issue is immediately apparent: a highlighted component of a difference graph may comprise more than one kind of difference. For example, in **diff-g2-g1**, the **Person** concept is highlighted; however, it may be highlighted for any of the following reasons:

- The **Person** concept doesn't appear in graph *g1* at all.
- The **Person** concept is a (possibly indirect) subtype or supertype of a concept that does appear in graph *g1*.
- The **Person** concept in *g1* may be an individual concept (i.e., one that has a referent).
- The **Person** concept "matches" a concept in *g1*, but *g2* uses a different canonical basis and therefore the two graphs aren't comparable via subtypes or supertypes at all.

Because of the different kinds of operations, we have found it useful to derive both difference sequences (i.e., one difference sequence and its inverse) so that we can use the coloring technique from the perspective of both the graphs, as shown in Figure 5. The complete list of operations appears in Figure 6. Bear in mind that the coloring operates for difference sequences in both directions.

Difference operation	Name	To be highlighted	Where
Specialize concept	**S-C**	Concept	Both graphs
Specialize relation	**S-R**	Relation	Both graphs
Generalize concept	**G-C**	Concept	Both graphs
Generalize relation	**G-R**	Relation	Both graphs
Insert subgraph	**I-G**	Added subgraph	Result graph
Delete subgraph	**D-G**	Deleted subgraph	Source graph
Insert co-referent link	**I-L**	Linked concepts	Result graph
Delete co-referent link	**D-L**	Un-linked concepts	Source graph
Move into context	**M-I**	Moved elements	Result graph
Move from context	**M-O**	Moved elements	Source graph

Figure 6. Coloring guidelines.

We have used the coloring technique in the example of the next section. This is our first attempt at visually describing graph differences; we expect that more elaborate and revealing color schemes may be possible in the future.

4 Applying the technique

One of the motivations of our work is that we wanted to develop techniques to identify differences between software development formal process models and models of the actual practice performed by developers. Conceptual graphs seemed a reasonable choice to represent both the process model and the practice model. In this

section, we illustrate the technique with a larger example, where we first calculate the difference sequence and then analyze it.

The graphs in this section were obtained in a practical modeling experiment where we were modeling an organization's requirements development process. The purpose of the two graphs is not central here; we will simply say that Figure 7 shows a process model of how the organization wants to operate, while Figure 9 shows how the organization actually operates.

4.1 Computing a difference sequence

We base the discussion of this section on the two graphs shown in Figure 7 and Figure 9. These graphs are the actual ones obtained in a realistic knowledge capture exercise with a software project manager.

One immediate reaction when inspecting *G1* and *G2* is that they seem intuitively quite similar. Since we used the same canonical basis for obtaining both graphs from our human knowledge source, this should not be surprising.

It should be noted that for large graphs, while some differences may be "obvious", there are many reasons why it may be difficult to find all of them manually: (i) we may draw the graph with different spatial arrangements, representing the same topology, (ii) some types may have similar names, and (iii) the sheer number of concepts and relations may be overwhelming for human perception.

To find a difference sequence for this pair of graphs, we follow the same procedure as before:

First, we found a least common generalization between the two graphs; this is not shown due to space limitations. Next we found two difference sequences between the LCG and the two graphs, which were then combined to show the difference sequence in Figure 8.

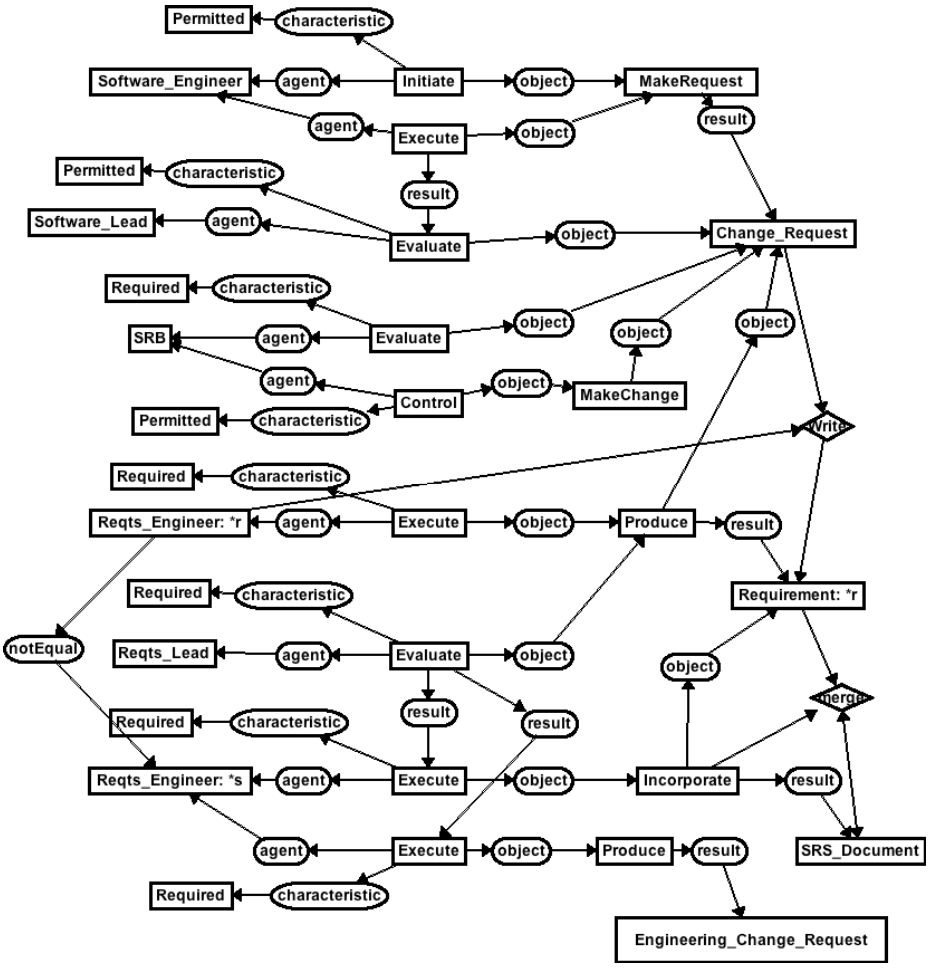


Figure 7. Example Graph *G1*, A Process Model.

DS(G1, G2) :

G-C: Generalize **Reqts_Engineer: *r** to **Reqs_Engineer**
G-C: Generalize **Reqts_Engineer: *s** to **Reqs_Engineer**
D-G: Delete **notEqual** relation
S-C: Specialize **Software_Engineer** to **Software_Engineer: Jerry**
S-C: Specialize **Reqts_Engineer** to **Reqts_Engineer: Jerry**
I-L: Insert **coreferent link** between **Software_Engineer: Jerry** and **Reqts_Engineer: Jerry** and **Reqts_Engineer: Jerry**

Figure 8. A difference sequence for two larger graphs.

We summarize the differences visually by highlighting them in the two graphs. Figure 10(a) shows a segment of Project-A's process model from Figure 7; Figure 10(b) shows a segment of its practice model from Figure 9. The white nodes are ones

where that model differs from the model to which it has been compared, i.e. it has a node the other model does not have, or the label of the node is different of the node in the same place in the other model; whereas the gray nodes are those that are the same for the two graphs.

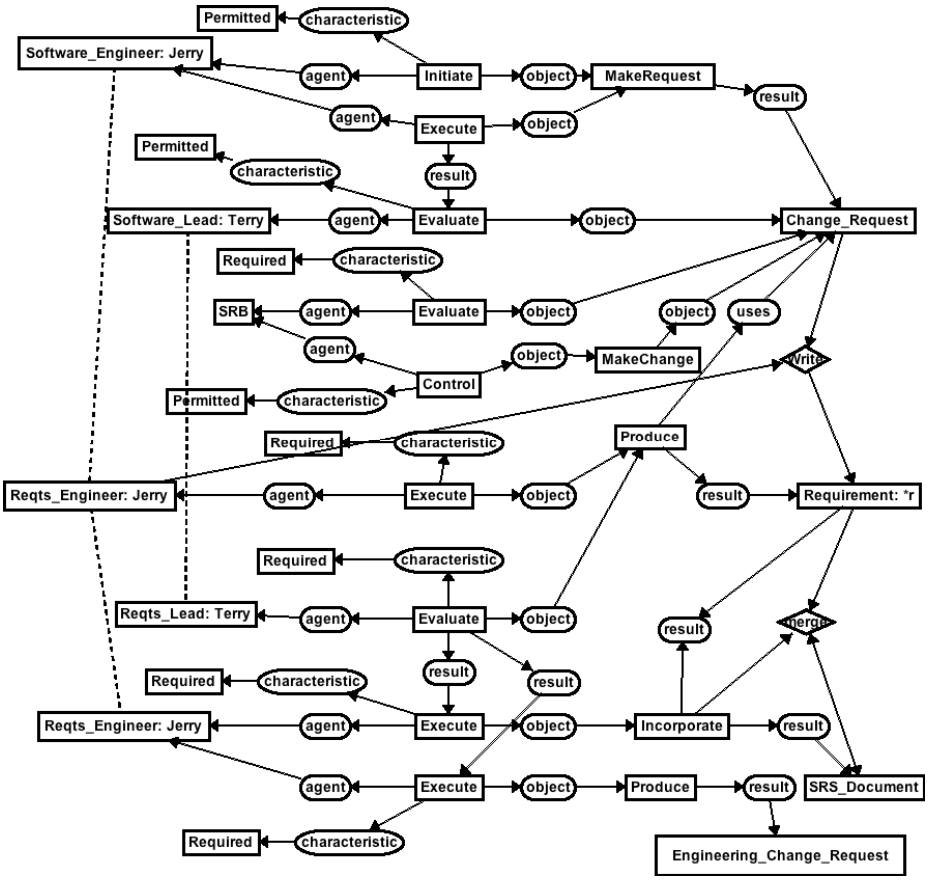


Figure 9. Example Graph *G2*, A Practice Model.

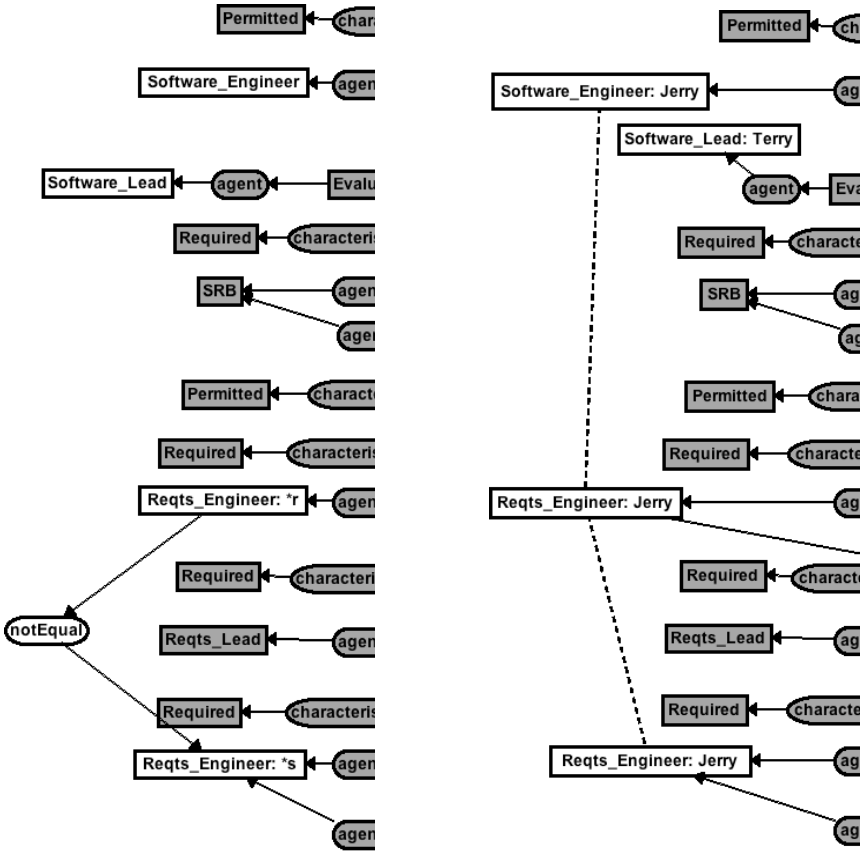
4.2 Interpreting the difference sequence

In our example, we determined two differences between the process model graph and the practice model as follows; these summarize what is shown in Figure 10.

- The process model contains no individual concepts, whereas the practice model shows individuals for some roles in the model. This reflects a natural tendency for practitioners to explain a process by identifying individual developers who are involved in the various roles.

The process model shows a **notEqual** relation between the requirements engineer who writes the requirement and the requirements engineer who incorporates the requirement into the SRS document. In the practice model, the requirements

engineer who produces the requirement (we call them ***r**) is also the same person (called ***s**) who incorporates the requirement into the SRS, a situation that is forbidden (through the **notEqual** relation) in the process model. The separation of roles (i.e., the explicit **notEqual** relation) in the process model represents an explicit prohibition, whereas in the practice model the separation between duties shown in the process model is lacking. Which model should be changed? Or should both be changed? In this final stage of the technique, it is up to actual participants such as the software managers and developers to interpret and analyze the comparisons.



(a) (b)

Figure 10. Highlighted Differences Between G1 and G2.

5 Discussion and Conclusion

For some readers, these kinds of operations take some getting used to, since they do not preserve truth characteristics. It is clear in our example that if we generalize **T:22** to **T** and then back to **T:34**, the two numbers are distinctly different and incompatible. As we mentioned earlier, our intent is to transform one graph into

another graph that is explicitly different; therefore they say different things and have different truth characteristics.

Based on the canonical graph set or existing definitions, some "differences" may in fact be merely expansions or contractions of definitions, while still preserving truth. For example, expanding a concept out to reveal its complete definition will in effect specialize a graph; however, there is no semantic change, since the definition was implied anyway by the original concept.

In general, for any given pair of graphs, there is no unique *LCG* for any pair of graphs. Among all the candidates, is one of them more useful in certain circumstances or optimal (i.e., more efficient) than the others? This is an open question. Since our purpose in developing the differences has been to support human analysis, we are satisfied for the moment with a possibly sub-optimal result with respect to its computation.

A difference sequence not only describes the differences but it also may help in measuring the degree of difference between the graphs; i.e., in general the more difference steps in the sequence, the greater the difference between the two graphs. Of course, if two graphs are completely different (i.e., their LCG is the graph **[T]**) then the difference sequence might consist of two steps – delete the first graph (a "subgraph") and then insert the second graph; which would only be two steps, yet the graphs would be completely different! The question of measuring the differences requires much more study.

One difficulty with this approach occurs if we try to compare two different graphs that were composed using differing canonical bases or ontologies. Transformations of one graph into another can therefore result in a graph that would be incorrectly formed ("nonsense") with respect to the original graph. Since we do not claim to preserve truth in performing these operations, this may not present a logical problem, but in some applications, differing ontologies may interfere with human interpretations of the results. These problems are well-known in ontology research (e.g., [6], [7]) and addressed by [8].

An interesting exercise would be to perform the graph difference algorithms on the actual canonical bases or ontologies themselves: this might lead us to discover some ontological differences between the knowledge bases they are intended to support.

There are important issues of tractability and computability which can be addressed more effectively if we relax the requirement that solutions be unique or deterministic. Of course, this means that two different analysts may derive two distinct difference sequences; our approach allows for possible multiple interpretations.

We developed our notion in order to use it in an application of conceptual graphs, not just as an interesting theoretical point. As a result, we have developed novel uses for color in conceptual graphs, in addition to demonstrating the usefulness of graph differences in aiding human analysis of conceptual graph models.

6 Bibliography

[1] E. Salvat and M.-L. Mugnier, "Sound and Complete Forward and Backward Chainings of Graph Rules," in *Conceptual Structures: Knowledge Representation as Interlingua*, vol. 1115, *Lecture Notes in Artificial*

Intelligence, P. W. Eklund, G. Ellis, and G. Mann, Eds.: Springer Verlag, 1996, pp. 248-262.
[2] D. Corbett, *Reasoning and Unification over Conceptual Graphs*. New York: Kluwer Academic, 2003.
[3] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*. Reading, Mass.: Addison-Wesley, 1984.
[4] G. Ellis, "Compiling Conceptual Graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, pp. 68-81, 1995.
[5] G. Ellis and F. Lehmann, "Exploiting the Induced Order on Type-Labeled Graphs for Fast Knowledge Retrieval," in *Conceptual Structures: Current Practices*, W. M. Tepfenhart, J. P. Dick, and J. F. Sowa, Eds. Berlin: Springer-Verlag, 1994, pp. 293-310.
[6] N. Guarino, "Formal Ontology, Conceptual Analysis and Knowledge Representation," *International Journal of Human-Computer Studies*, vol. 43, 1995.
[7] A. Farquhar, R. Fikes, W. Pratt, and J. Rice, "Collaborative Ontology Construction for Information Integration," Knowledge Sharing Laboratory, Stanford University, Technical Report KSL-95-63, August 1995 1995.
[8] M.-L. Mugnier and M. Chein, "Characterization and algorithmic recognition of canonical conceptual graphs," in *Conceptual Graphs for Knowledge Representation*, vol. 699, G. W. Mineau, B. Moulin, and J. F. Sowa, Eds. Berlin: Springer-Verlag, 1993, pp. 294-311.