Towards a User-Friendly Semantic Formalism for Natural Language Generation

by Craig Thomas

Abstract

Computational semantics has become an interesting and important branch of computational linguistics. Born from the fusion of formal semantics and computer science, it is concerned with the automated processing of meaning associated with natural language expressions [2]. Systems of semantic representation, hereafter referred to as semantic formalisms, exist to describe meaning underlying natural language expressions. To date, several formalisms have been defined by researchers from a number of diverse disciplines including philosophy, logic, psychology and linguistics. These formalisms have a number of different applications in the realm of computer science. For example, in machine translation a sentence could be parsed and translated into a series of semantic expressions, which could then be used to generate an utterance with the same meaning in a different language [14]. This paper presents two existing formalisms and examines their user-friendliness. Additionally, a new form of semantic representation is proposed with wide coverage and user-friendliness suitable for a computational linguist.

Introduction

Semantic formalisms are becoming an important part of many applications in computational linguistics. Regardless of the task, a semantic formalism needs to be sufficiently expressive to capture subtle variations in meaning. Expressiveness is evaluated based on a number of different criteria. For example, to be useful for a diverse number of applications, a formalism must have wide linguistic coverage and be precise. Wide coverage means the formalism can express a large number of linguistic phenomena, from simple predicates to more complex items including modality, tense, and quantification. Precision means the formalism can provide a distinction between apparently similar, but semantically different phenomena. For example, the differences between an inclusive and exclusive *or*. The question "Would you like cream or sugar?" demonstrates an *inclusive or*; it implies none, one, or both cream or sugar as an option. The question "Would you like coffee or tea?" is exclusive; it implies either one or the other, but not both. In this case, precision may be accomplished by using different symbols or operators.

Unfortunately, one major aspect of usability is missing from this evaluation, namely the degree of user-friendliness. While user friendliness is not important for machines or automated tasks, it is crucial for humans. In these instances, it is vital to have a readable, understandable, and intuitive formalism.

Consider three different situations. Imagine that a reporter is attending a hockey tournament and wishes to record game details in a clear and precise manner. Imagine also a children's book author is writing a narrative for a fairy tale. Finally, imagine a university student is writing a report for a class. As an additional constraint, imagine that all three of these publications need to be printed in both English and French. Ideally, the information need only be recorded once in a language independent fashion using a semantic formalism. Once recorded, the information could be fed to a natural language generation system that would print the same story in both English and French. The flow of this process is presented graphically in Figure 1. It will be best if the authors can use an intuitive formalism without becoming a linguistic expert or logician. The question is whether such user-friendly semantic representation system already exists

The purpose of this paper is two-fold. Firstly, to present two existing semantic formalisms: the first order predicate calculus and Montague's intensional logic. The basic machinery of each are presented with simple linguistic examples, while more complex examples are used to demonstrate their limited degree of user-friendliness. Secondly, it will present and discuss some preliminary results from a new system of semantic representation. Few of the goals of this new system are to provide wide coverage, and to be user-friendly, therefore suitable for

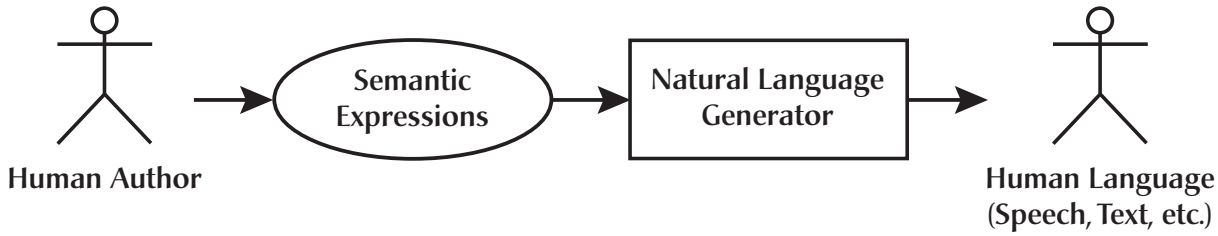

Figure 1: A flowchart depicting the use of human written semantic expressions in a natural language generation task.

computer scientists and computational linguists. The remainder of this paper will proceed as follows. The next section examines both the first order predicate calculus and Montague's intensional logic, and demonstrates their use and degree of user-friendliness. The following section presents the form of the new system of semantic representation and some linguistic examples demonstrating its use. Next, areas for future work are covered. Finally, the last section provides a brief summary and concluding remarks.

Evaluating User-Friendliness

Complexity issues tend to cause formalisms to be less user-friendly. This complexity typically manifests itself in one of two ways. The formalism is based upon a model that appears to be straightforward and easy to apply to a variety of linguistic phenomena. There are many formalisms available that appear to behave intuitively, for example, the first order predicate calculus (usually referred to as first order logic). A second instance of complexity arises if the semantic formalism is based upon an extremely intricate model. More detailed models are needed to elegantly express complex linguistic phenomena. An example is Montague's intensional logic.

First order logic is commonly used for representing natural language expressions, partly because of the abundance of tools available for inferencing tasks and theorem proving [2]. Recall that first order logic has a model **M = [A, F]** which represents the world, against which all expressions are evaluated. Within **M** there is the domain of individuals **A**, and an assignment function **F** [3]. As an example, **A** will contain the individuals *Marcus Junius Brutus, Julius Caesar* and *Juliet Capulet*. The assignment function **F** is responsible for mapping syntactic names and predicates to semantic values. In our example, the term **brutus** maps to *Marcus Junius Brutus*, while the predicate **MALE** may have the mapping: (*Marcus Junius Brutus* → 1, *Julius Caesar* → 1, *Juliet Capulet* → 0). The overall truth-value of an expression is obtained by evaluating it relative to the model **M**. For example, the truth-value of the expression:

```
MALE(brutus)
```

would be true if the predicate **MALE** adheres to the mapping above, and if brutus maps to *Marcus Junius Brutus*. First order logic is model-theoretic and truth-theoretic in nature as it uses a model to reflect some reality. The user obtains the meaning of an expression by evaluating a series of truth conditions [3]. This approach falls under denotational semantics, since the meaning of a sentence depends on the value that is obtained through the evaluation of expressions [19]. Although it appears that first order logic is user-friendly, when we attempt to represent more complex linguistic phenomena, the thin veneer of simplicity begins to melt away, as demonstrated later.

Montague's intensional logic goes further than first order logic by introducing world and time coordinates directly to the underlying model, as well as defining a rich set of operators to deal with extensionality and intensionality [3]. To illustrate, the intensional logic model is **M = [A, W, T, <, F]** where **A** and **F** were as in first order logic, **W** is a set of possible worlds, **T** is a set of instances of times, and **<** is a linear ordering on **T**. I will explain these domains in a later section. For now it is enough to know that these types of formalisms typically have expressions that are elegant compared to their first order counterparts. Unfortunately, this elegance comes with a price to the user, who needs detailed and expert knowledge of the underlying model and operational symbols.

"Simple" Statements

To demonstrate how first order logic becomes cumbersome, we can examine some relatively "simple" English statements. Consider the sentence "Brutus is a man." Expressing this utterance in first order logic is straightforward:

```
MALE(brutus)
```

Where is the user-unfriendliness of the formalism? Consider the sentence "Brutus is an honorable man" expressed below:

```
MALE(brutus) & HONORABLE(brutus)
```

Here are the first signs of trouble. The logical connective & is embedded in the semantic expression despite the fact that the word *and* is not present in the English sentence [7]. Our translation from the semantic expression to an English utterance is more analogous to "Brutus is a man and Brutus is honorable" than "Brutus is an honorable man." This subtle difference makes it appear as if we have lost a degree of precision with our translation. Although many would write this anomaly off as a simple consequence of translation from English to a logical language, more problems begin to surface if we press further. Consider the expression "Brutus stabs Caesar to death":

```
STABTODEATH(brutus, caesar)
```

Unfortunately, this particular expression suffers from reduced generality, since the actions **STAB** and **DIE** become fused into the predicate **STABTODEATH**. A more natural tendency would be to create an expression such as:

```
STAB(brutus, caesar) & DIE(caesar)
```

However, the logical & operator fails to express the implication of **STAB** in relation to **DIE** [7]. A literal translation of the semantic expression yields "Brutus stabs Caesar and Caesar dies." Although the English description of the logical formula *appears* to link **STAB** to **DIE**, there is no *causal connection* between the two predicates. For example, the same semantic structure occurs in the sentence "Tom drives the car and the sky is gray" below:

```
DRIVE(tom, car) & GRAY(sky)
```

As is intuitively obvious, the fragment "Tom drives the car" in no way implies "the sky is gray." Using this same structure, we conclude that "Brutus stabs Caesar" in no way implies "Caesar dies." Using material implication gets us closer to a proper meaning:

```
STAB(brutus,caesar) → DIE(caesar)
```

However, the direct translation becomes "if Brutus stabs Caesar then Caesar dies," which is not quite representative of the action, since the cause of Caesar's death is not conditional. It becomes more complex with "Brutus stabs Caesar with a knife":

```
∃x( KNIFE(x) & STAB(brutus, caesar, x) )
```

Read literally, the expression states "there exists some thing **x**, such that **x** is a member of the set of things that are knives, and Brutus stabs Caesar with **x**." An existentially quantified **x** is required

to represent the indefinite description of "a knife", which signals that there is an object known to be a member of the set of knives, but no uniquely identifiable knife is singled out. Even more complicated would be to define the causative agent of Caesar's death in "Brutus stabs Caesar to death with a knife":

```
∃x( KNIFE(x) & STABTODEATH(brutus, caesar, x) )
```

In **STABTODEATH**, it is unclear whether Brutus, the knife, or the act of stabbing Caesar with a knife is the causative agent in Caesar's death.

From these examples, it is apparent that the correspondence between English and first order logic is fraught with difficulties. So called "simple" sentences result in semantic constructions which bear no relation to their English counterparts, making first order logic a non user-friendly system for the average user. This gets worse when more complex linguistic phenomena come into play.

Tensing Up

So far, we have examined all linguistic examples that have occurred in the present tense. With English syntax, the lack of "tense operators" signals that interpretation of the expression is understood to occur at the current time [16]. In order to represent tense, we need a model that incorporates instances of time, so that we can specify whether an action occurs in the past, the present or the future. With first order logic, we can add instances of time and events to our domain **A** as suggested by [2]. For example, "Caesar died" would become:

```
∃e( DIE(caesar, e) & t( TLOC(e, t) & t < s) )
```

The expression translates to "there exists some event **e** that is the event of Caesar's death, and there exists some time **t** that is the time of event **e** (**TLOC**), and **t** occurs before **s** (the current time)." Now we can see how the first order logic is unfriendly. Domain **A** was expanded to include different types, namely instances of time and events [2]. While this change is not devastating, it does introduce a degree of complexity that we didn't need to account for before. For example, the **MALE** predicate applied only to individuals. How does **MALE** apply to events or times? Since we don't differentiate based on types in first order logic (i.e., we don't differentiate between types of things in our domain of individuals so *Marcus Junius Brutus*, a specific time, and a specific event are all the same type), we now have to be careful to make sure our semantic expressions are formed properly.

Additionally, two existentially quantified variables, a specialized predicate **TLOC** and knowledge of the **s** variable are needed to interpret this semantic statement. All of these issues make the expression more difficult to understand. Fortunately, Montague's intensional logic representation offers relief:

```
P[ die'(caesar') ]
```

The main predicate **die'(caesar')** in intensional logic is interpreted the same way as the first order logic predicate would be. Recall that instances of time **T** and a linear ordering on **T** are built into the model for intensional logic. The **P** operator exploits the linear ordering on **T** by forcing the evaluation of the predicate **die'(caesar')** to occur at some time coordinate **t'** where **t'** is before the current time **t**. In other words, the predicate will be true if **die'(caesar')** is true at any point in time **t'** such that **t' < t** [3]. This notation provides us with a way of expressing the past tense more elegantly than first order logic.

Unfortunately, the model of tense presented here is incomplete. Tense combines with grammatical aspect, which describes the nature of an action over time. This combination of aspect with tense requires changes to the simple model described above. In particular, to achieve a precise differentiation between the progressive aspect, such as the expression "Brutus was stabbing Caesar", and the perfect aspect, such as the expression "Brutus had stabbed Caesar", the model needs to include a reference time called **r** [17]. The theory behind introducing such a reference time **r** is that linguistic tense involves more than just knowing what time it is "now" and the time that an event occurred. If the action took place before reference time **r**, and has not yet been completed, then the action is continuing (progressive aspect). If the action occurs at the same time as **r**, then the action is complete (perfect aspect). The reference time **r** in relation to the current time **s** tells us if this action is in the past, present or future [17]. For example, the expression "Caesar had gone to the senate", in first order logic becomes:

```
∃e( GO(caesar, senate, e) & ∃t( TLOC(e, t)
   & t < r & r < s) )
```

The literal reading of this statement is: "there exists an event e known as Caesar going to the Senate, that took place at time **t**, which exists at some temporal point prior to some reference time **r**, and in turn **r** occurs prior to **s** (the current time)." This ordering on time provides us the reading of "had gone" by suggesting that the action was complete before time **r**, and that **r** occurred at some point in the past. Thus we get the past perfect reading. With intensional logic, the same sentence is represented by the expression:

```
PP[ go'(caesar', senate') ]
```

While the intensional logic representation is still elegant, the repeated use of the **P** operator adds new complications during interpretation. However, this is not the main issue; the problem is that the model differentiates only between two aspects and does nothing for the multitude of aspects that may exist in other languages. In conclusion, although the intensional formalism is elegant, it lacks the necessary coverage required to deal with all of the possible tenses and aspects. From previous discussions, it is obvious that first order logic lacks coverage as well.

Indefinite Problems

Thus far, intensional logic appears to be user-friendly. To demonstrate how it can become unfriendly, consider the sentence "Caesar is searching for a conspirator", which is structurally identical to Montague's own example of this phenomenon [3, 13]. With this particular sentence, we get two different readings of "a conspirator": specific and non-specific readings. In one instance, we can imagine that there is a specific conspirator that Caesar is looking for (i.e. Brutus). This reading is relatively simple to express with intensional logic, and is expressed in first order logic in exactly the same way:

```
∃x[ conspirator'(x) & searchfor'(caesar', x) ]
```

In the second, non-specific instance of "a conspirator", Caesar believes that there is conspirator, but in reality none exist. This reading poses a problem for first order logic, as the existential quantifier states that at least one conspirator exists in reality. However, since there is no entity in the actual world, we cannot capture Caesar's

actions in a concise manner. Intensional logic solves this problem, but does so by revealing its unfriendly nature:

```
searchfor'( caesar', ^λQ∃x[ conspirator'(x)
   & ᵛQ(x) ] )
```

The expression above states that Caesar is seeking something with conspirator-like properties [3]. To understand this reading, we must discuss the semantics of the **^** operator, otherwise known as the intensional operator. This operator forces evaluation of the statement to occur across all points in time and across all possible worlds [3]. In intensional logic, each predicate and constant term may have a different value at a different time, or another possible world coordinate. Thus, while **bald'** may have a mapping of (*Marcus Junius Brutus* → 0, *Julius Caesar* → 0, *Juliet Capulet* → 0) at coordinates **(t0, w0)**, at coordinate **(t1, w0)** it may be (*Marcus Junius Brutus* → 1, *Julius Caesar* → 1, *Juliet Capulet* → 0) [3]. If we add the **^** operator to **bald'(brutus)**, the truth-value will evaluate to true if **bald'(brutus)** is true at any time or world coordinate [3]. The overall effect is that **conspirator'(x)** is evaluated in any possible world or time. This allows us to specify an individual that may not exist in this reality.

The **ᵛ** operator forces evaluation of the predicate **Q** to take place at the current world and time coordinate [3]. When combined with **^λ**, it creates a grounding to the actual world that says that **x** has the properties of being a conspirator, regardless of whether one exists here and now. The process of describing the set of properties that define what a conspirator is acts as a way of specifying what Caesar is seeking without having to indicate a specific individual from the actual world [3]. In other words, it allows us to specify a "thing" as an abstract set of properties without "it" having to exist in any reality. As we can see, the interpretation of such statements in the intensional logic requires a great deal of knowledge and expertise. Thus, I claim that this formalism is unfriendly.

Other Formalisms

Other formalisms not yet mentioned include Conceptual Structures [7, 8], Case Grammar [6], Discourse Representation Theory [5], Functional Unification Grammar [11, 10], Systemic Functional Grammar [20], Conceptual Dependency [18], and Semantic Networks [12]. Although the scope of this paper does not allow for the inclusion of explicit examples of each of these semantic formalisms, I acknowledge that there are benefits and drawbacks to each system, whether they have been computationally realized or not. That being said, I would claim that all these formalisms suffer from reduced user-friendliness at the expense of expressive power. However, readers are encouraged to consult the sources of these other systems and formulate their own opinions based on their findings.

A Simpler System

The purpose of our research is to create a precise semantic formalism with wide linguistic coverage specifically designed for use by non-expert computational linguists for the purposes of natural language generation. In the traditions of first order logic and intensional logic, semantic expressions have to be unambiguous, and able to precisely describe the contents of any natural language utterance. We are in the process of developing a transformational algebra to account for why two semantic expressions may yield the same meaning, but have different forms. The particular formalism presented here was first proposed by Levison & Lessard [13], further explored by Donald [4] and is being formally defined and developed to much more detail by this research.

The evaluation of an expression's "meaning" will be accomplished quite differently than first order logic and intensional logic. Rather than relying on a model and truth-values, our formalism will use functions that have atomic meaning. Meaning isn't achieved through the denotational value of a statement, but rather through the combination of atomic blocks of meaning in complex ways. In this aspect, our atomic statements are similar to Jackendoff's notion of concepts [8]. Our functions literally *mean* the information they are meant to convey. To obtain expression meaning, a user needs to apply the functions to argument variables and their overall structural configurations. Furthermore, by utilizing functions, our formalism will be similar to a functional programming language such as Haskell [1].

Expressions in our formalism are composed of functions with fixed valencies that require parameters of various semantic types [4]. These semantic types correspond to semantic notions such as: completions (full thoughts), entities, qualities, circumstances and actions. For example, a function corresponding to the action *stab* would have a return type of *completion* and takes two *entities* as arguments. Special types of functions, *zero-valent* functions, are known as atomic parameters and may represent semantic individuals. For example, **brutus**, **caesar** and **juliet** are all zero-valent functions that represent the individuals *Marcus Junius Brutus*, *Julius Caesar* and *Juliet Capulet* accordingly. All functions, both atomic and multi-valent, are contained within a semantic lexicon, which contains information relating to their valency, parameter types and return types, encyclopedic meaning and inferences possible under various contexts. For example, "Brutus buys a knife from Cassius" entails the facts that the knife exchanged ownership from Cassius to Brutus, and a monetary exchange occurred from Brutus to Cassius. This information would be captured in the semantic lexicon. A list of valid modal and temporal contexts would also be included, because the modal context *wish* in "Brutus wishes to buy a knife from Cassius" does not entail the fact that any exchange actually took place. Putting this all together, to express the statement "Brutus stabs Caesar" one may simply write the following expression:

```
stab(brutus, caesar)
```

The function above relates entities **brutus** and **caesar** in a *stab* relation, with **brutus** being the stabber and **caesar** being stabbed. It is worthwhile to note however, that the form of the semantic expression may change, depending on how the programmer wishes to organize and define functions within the lexicon. For example, the following semantic expression might be used instead of the one above:

```
vtd(stab, brutus, caesar)
```

Here, our preference was to create an overarching function called **vtd** whose function is to relate the entity *brutus* performing a transitive *stab* action to the direct object entity *caesar*. The details of the how each function appears in the semantic lexicon are left up to the programmer. It is worthwhile to note however, that the following two functions are not equivalent:

```
stab(brutus, caesar)
stabr(brutus, caesar, knife[indefinite])
```

While both functions above relate the fact that Brutus stabs Caesar, the function **stabr** carries a third argument reserved for an *instrument*. If read literally, it states that "Brutus stabs Caesar with a knife". Rather than having two separate functions which are semantically related to each other, we could simply have **stabr** and use **unspec** in the third argument place. By definition, **unspec** is a multi-typed constant which is used to denote the fact that the given parameter is to be left *unspecified*. The following would be equivalent to **stab(brutus, caesar)**:

```
stabr(brutus, caesar, unspec)
```

The **unspec** constant can be placed in other argument locations giving rise to a multitude of meanings. For example, "Caesar is stabbed with a knife":

```
stabr(unspec, caesar, knife[indefinite])
```

Or "Brutus stabs someone or something with a knife" below. We can prevent ambiguity of specifying someone or something by placing a semantic restriction on the argument type in the funtion definition of **stabr**. For the sake of brevity, this discussion is omitted.

```
stabr(brutus, unspec, knife[indefinite])
```

The system as described so far looks nearly identical to first order logic, and may leave readers asking, "What is so special about it?" Some of the power of our formalism comes in the form of adjustments that can be made to any function. Adjustments are *bracketed attachments* which may contain information relating to aspect, modality, tense, etc. We have already seen these attachments in our **stabr** function where they were used to indicate the indefiniteness of "a knife". Similar notation can be used to indicate tense. For example, "Brutus stabbed Caesar":

```
stab[past](brutus, caesar)
```

Or for the completed past sentence "Brutus had stabbed Caesar.":

```
stab[past,complete](brutus, caesar)
```

Not to be overlooked is the fact that functions themselves may be passed as arguments, assuming their types match the argument places they are being inserted into. For example, "Brutus stabs Caesar with a red knife":

```
stabr(brutus, caesar,
   qual(knife[indefinite], red))
```

The **qual** function takes an *entity* and a *quality* as inputs and returns an *entity*. In this particular case, the indefinite knife is qualified as being red. For ease of use, a **qlist** function is available that returns a concatenated list of qualities associated with an item. For example, to indicate that the knife is red and blue, one may simply write:

```
qual(knife, qlist(red, blue))
```

In fact, the underlying mechanism performing the concatenation (**cat**) in the **qlist** may be generalized to work with any function, allowing us to create list of entities (**elist**), lists of actions (**alist**), etc. Thus a list of entities such as "Caesar, Brutus and Cassius" would be written with an elist:

```
elist(caesar, brutus, cassius)
```

One limitation of first order logic can be addressed by our formalism because the user can create interrogative sentences using a special *query* function (or more simply **?**). Perhaps the user wishes to ask "who (or what) stabbed Caesar?"

```
stab(?, caesar)
```

Perhaps the user wishes to know the instrument that Brutus used to stab Caesar with in "What did Brutus use to stab Caesar?":

```
stabr[past](brutus, caesar, ?)
```

Or "what color was the knife that Brutus used to stab Caesar?":

```
stabr[past](brutus, caesar,
   qual(knife, color?))
```

Many other aspects of a language can be represented using this formalism. As our investigation continues, we are developing means of representing phenomena such as quantification, sequences, lists, conjunction, disjunction, relative clauses, anaphora, multi-utterance discourse, etc.

To put it all together, consider some fragments from the well known fairy tale of the Three Little Pigs [9], represented using our semantic expressions:

```
/* Declarations of the pigs, the mother and
   the wolf */
entity pig1 = qual(pig[spec], qlist(male,
   little));
entity pig2 = qual(pig[spec], qlist(male,
   little));
entity pig3 = qual(pig[spec], qlist(male,
   little));
entity threepigs = elist(pig1, pig2, pig3);
entity motherpig = qual(pig[spec],
   qlist(female, mother_of(REL, threepigs)));
entity wolf = qual(wolf[spec], male);

/* Once upon a time, there was a mother pig that
   had three sons. */
have[past](motherpig, threepigs);

/* She did not have enough money to support them,
   so she sent them out to build their own houses. */
cause(
        have_for[not](oldsow, money,
                care_for(threepigs)),
        cause(
                oldsow,
                alist(
                        build(pig1, house),
                        build(pig2, house),
                        build(pig3, house)
                )
        )
);

/* The first little pig built a house made out
   of straw. */
```

```
entity strawhouse = house[indefinite];
build_using(pig1, strawhouse, straw);

/* One day, a wolf came and knocked on the door. */
sequence(
        go(wolf, strawhouse),
        knock_on(wolf, door_of(strawhouse))
);

/* The wolf said "little pig, let me in." */
say(
        wolf,
        pig1,
        cause(pig1, enter(wolf, strawhouse))
);

/* The first little pig said "not by the hair
   of my chinny chin chin." */
say(
        pig1,
        wolf,
        cause[not](pig1, enter(wolf, strawhouse))
);

/* The wolf said "then I'll huff, and I'll puff
   and I'll blow your house down." */
say(
        wolf,
        pig1,
        cause(
                wolf,
                cause(
                        blow_on(wolf, strawhouse),
                        destroy(wolf, strawhouse)
                )
        )
);
...
```

Additionally, an example portion of the semantic lexicon is included:

```
/*
   function name :: type signature
   | natural language description
   | encyclopaedic knowledge
   |(functions describing this function)
   | semantic transformations (implications)
   #
*/

blow_on :: entity en1, entity en2 -> act
| "(en1) blows on (en2)"
|
|
#

broken :: quality
| "is non-functional or destroyed"
|
|
#

build :: entity en1, entity en2 -> act
| "(en1) builds (en2)"
|
| create(en1, en2); exist(en2);
#

build_using :: entity en1, entity en2,
entity en3 -> completion
| "(en1) builds (en2) using (en3) as
| the raw material"
|
| create(en1, en2); made_of(en1, en3);
| exist(en2);
#

care_for :: entity en1, entity en2 -> act
| "(en1) spends time and resources to take care
| of the well being of (en2)"
|
|
#

cause :: entity en1, act act1 -> completion
| "(en1) causes (act1) to occur"
|
|
#

destroy :: entity en1, entity en2 -> act
| "(en1) destroys (en2)"
|
| qual(en2, broken)
#

door_of :: entity en1 -> entity
| "a door belonging to (en1)"
| part_of(this, en1)
|
#

...
```

Future Work

From the examples given, we hope to impress upon the reader that this system is much more natural from a nonexpert linguist or logician point of view, and offers greater flexibility and power than any other system. However, while we provide theoretic information and concrete examples as to how the semantic formalism functions, there is much work yet to be done.

In particular, the notions of linguistic coverage and user-friendliness need to be addressed. First, to ensure wide coverage, a number of linguistic examples are being gathered from a number of linguistic phenomena. The inner workings of each example will be investigated to determine if additional mechanisms are needed for their expression. A preliminary list of phenomena includes: declaratives, imperatives, interrogatives, negation, conjunction, disjunction, sim-

ple tense (past, present, future), aspect (indefinite, complete, continuing, etc.), logical quantifiers (every, one, none), non-logical quantifiers (many, most, some, few, etc.), multi-sentence discourse, modalities, beliefs, generics, anaphora, speech acts, meronymy, synonymy, hyponymy, polysemy, and others.

Secondly, the notion of user-friendliness must be addressed more precisely. While no evaluation methodology has yet been determined, several options are being considered. One option is users evaluation, where a selection of computer scientists would become users, and would be asked to formulate a number of meanings. Time taken to formulate a response, its accuracy with respect to the intended meaning, and general user observations would all become evaluation criteria. The task could also proceed in reverse: users would be given a semantic expression and asked to summarize its meaning using natural language. Again, time, accuracy, and user observations would become evaluation criteria. Another option is a study involving expression of complex narratives, where the number of semantic primitives introduced is used to measure complexity. Too many primitives make a cumbersome formalism; the extreme case being a primitive for each completion. Too few primitives would also exert a negative effect by adding a layer of notational complexity for complex phenomena and possibly impacting expressiveness. Here, the ratio of completions to primitives would provide insight into user-friendliness. Regardless of the evaluation methodology, a clear and precise metric is needed.

Finally, since our goal is to map this formalism onto a full generation environment, the actual process of transforming semantic expressions into natural language utterances must also be explored.

Conclusions

No current system is truly user-friendly. While first order logic and Montague's intensional logic are being used to express a number of linguistic phenomena, they do so at the expense of creating complexity for the user. Unfortunately, this complexity continues to grow as more complex linguistic phenomena are explored.

This paper explored the form of a new system of semantic representation which has considerable expressive power, while maintaining a more intuitive form to its users. While many feel a formalism's expressive power is more important than user friendliness, developing an intuitive formalism could lead to wide-spread use in natural language generation tasks and possibly even to the larger natural language processing community.

Indeed, to verify any automated task, a human being must be present to review and interpret any semantic intermediate form created during the debugging process. Simply put, the simpler the semantic form becomes, the easier the verification process will be. In terms of natural language generation, this formalism coupled with a natural language system such as VINCI [13] could develop into a new and simple way of providing a wide range of users the ability to generate multi-language utterances with a minimal amount of effort.

References

1. Bird, R. 1988. *Introduction to Functional Programming Using Haskell* 2nd Ed. Prentice Hall Series in Computer Science. Prentice Hall.
2. Blackburn, P. and Bos, J. 2003. Computational semantics. *Theoria 18*, 46. 27-45.
3. Dowty, D. R. 1981. *Introduction to Montague Semantics*. Synthese Language Library 11. D. Reidel Publishing Co., Dordrecht, Holland.
4. Donald, M. C. 2005. A metalinguistic framework for specifying generative semantics. Master's thesis, Queen's University.
5. van Eijck, J. 2006. Discourse representation theory. In *Encyclopedia of Language & Linguistics*, K. Brown, Ed. Elsevier. 660-668.
6. Fillmore, C. J. 1968. The case for case. In *Universals in Linguistic Theory*, E. Bach & R. T. Harms, Eds. Holt, Reinhart and Winston, New York, NY. 1-90.
7. Jackendoff, R. 1983. *Semantics and Cognition*. Current Studies in Linguistics Series. The MIT Press, Cambridge, MA.
8. Jackendoff, R. 1990. *Semantic Structures*. Current Studies in Linguistics Series. The MIT Press, Cambridge, MA.
9. Jacobs, J. 1987. The story of the three little pigs. In *Classics of Children's Literature* 2nd Ed., J. W. Griffith and C. H. Frey, Eds. Macmillan Publishing Co. 815-817.
10. Kasper, R. T. and Rounds, W. C. 1986. A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, Morristown, NJ. 257-266.
11. Kay, M. 1984. Functional unification grammar: A formalism for machine translation. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting on Association for Computational Linguistics* (*ACL-22*). Association for Computational Linguistics, Morristown, NJ. 75-78.
12. Lehmann, F. 1992. Semantic networks. *Comput. Math. Appl. 23*, 2-5. 1-50.
13. Levison, M. and Lessard, G. 2004. Generated narratives for computer-aided language teaching. In *eLearning for Computational Linguistics and Computational Linguistics for eLearning* (*COLING'04*), Geneva, Switzerland. L. Lemnitzer, D. Meurers, and E. Hinrichs, Eds. 26-31.
14. Lønning, J. T., and Oepen, S. 2006. Re-usable tools for precision machine translation. In *Proceedings of the COLING/ACL Conference on Interactive Presentation Sessions*. Association for Computational Linguistics, Morristown, NJ. 53-56.
15. Montague, R. 1974. The proper treatment of quantification in ordinary English. In *Formal Philosophy*, R. Thomason, Ed. Yale University Press, New Haven, CT.
16. Moss, L. S., and Tiede, H.-J. 2006. Applications of modal logic in linguistics. In *Handbook of Modal Logic*, P. Blackburn, J. F. van Benthem, and F. Wolter, Eds. Elsevier Science. 1031-1076.
17. Reichenbach, H. 2004. The tenses of verbs. In *Semantics: A Reader*, S. Davis and B. S. Gillon, Eds. Oxford University Press, Oxford, UK. 526-534.
18. Schank, R. C. 1972. Conceptual dependency: A theory of natural language understanding. *Cognitive Psych. 3*, 4. 552-631.
19. Tennent, R. 1991. *Semantics of Programming Languages*. Prentice Hall International Series in Computer Science. Prentice Hall International, New York, NY.
20. Winograd, T. 1982. *Language as a Cognitive Process: Syntax*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Biography

*Craig Thomas (**craig@craigthomas.ca**) is a PhD candidate in the Computational Linguistics Laboratory in the School of Computing at Queen's University in Kingston, Ontario, Canada.* *This research was conducted with funding assistance from the Natural Sciences and Engineering Research Council of Canada.*