Semantic Inference by First Order Logic

Zhuoyun Ao and Dale Lambert
Command Control Communication and Intelligence Division
Defence Science Technology Organisation
South Australia, Australia
Email: {Zhuoyun.Ao,Dale.Lambert}@dsto.defence.gov.au

Abstract—Higher-level fusion involves machine based assessments of situations in the world, with situations ultimately being represented by sets of propositions. To facilitate fusion, a common canonical language is required to represent these propositions within the machine and to perform appropriate inferences with them. This paper presents an implemented first order logic programming-based approach to this problem, while incorporating novel improvements relating to structure preserving formula renaming and Skolem function elimination. We prove that both of the introduced techniques are truth preserving. Experimental results show that the implementation is able to not only generate structure preserving normal clauses, but also avoids an exponential increase in number of clauses.

Keywords: theory representation; higher-level fusion; situation assessment; formula-renaming; clausal normal form; logic-programming; meta-interpreter.

I. INTRODUCTION

The State Transition Data Fusion (STDF) Model ([1]) refinement of the JDL Data Fusion Model (e.g. [2]) understands the world through four levels of fusion processing.

• Level 0 fusion represents observables in the world by transitions (sequences) of feature vector state estimates within the machine. Each feature vector state estimate is typically a numerical measurement vector of an observable’s measurable features formed from signal, textual or image processing of a single sensor sampling of the observable.

• Level 1 fusion represents objects in the world by transitions (sequences) of state vector estimates within the machine. Each state vector estimate is typically a numerical measurement vector of an object’s measurable properties formed from signal, textual or image processing of a sequence of feature vector state estimates of an observable for that object.

• Level 2 fusion represents situations in the world by transitions (sequences) of states of affairs estimates within the machine. Each state of affairs estimate is typically a set of symbolic propositions about the world, formed from the inferential extension of a set of perceptual atomic propositions, with each perceptual atomic proposition being a relation symbol applied to a sequence of object symbols. Each object symbol in each perceptual atomic proposition represents an identity formed from processing a sequence of state vector estimates for an object or a measurable property value for that identity, while the relation symbol in each atomic proposition is deduced from the measurable property values of that identity or identities.

• Level 3 fusion represents scenarios in the world by transitions (sequences) of scenario state estimates within the machine. Each scenario state estimate is typically a sequence of sets of symbolic propositions about the world formed as a situation estimate from a sequence of states of affairs estimates of a scenario in the world.

The STDF model also holds that the generic fusion process identified in Figure 1 applies contextually to each of these four levels. Level 2 registration begins the focus of this paper. Registration at level k translates the outputs of the detection process at level k into a common canonical format so that fusion of the detection information at level k becomes possible. For example,

- The level 1 detection processes are the level 0 processes. Thus level 1 registration is usually coordinate registration for subsequent data association in the case of signal and image processing, and word registration for subsequent lexical association in the case of text processing ([1]).
- The level 2 detection processes are the level 1 processes. Thus level 2 registration is usually semantic registration to form perceptions for subsequent propositional association with predicted expectations ([1]).

The common canonical symbolic language for level 2 fusion needs to be able to represent perceptual aspects of the world for semantic registration. To represent more abstract states of affairs of interest, it also needs to be able to semantically...
represent more abstract concepts than perceptual concepts. For example, we do not directly perceive another’s beliefs or authority. These are instead inferred from perceptions satisfying expectations. The canonical symbolic language therefore needs to cater for a broad range of concepts and do so semantically, both in the sense that we humans understand what the concept symbols mean, and in the sense that the machine respects those meanings when conducting inferences with those symbols. This paper addresses some of the technical details associated with an approach to a canonical semantic symbolic language for level 2 fusion.

II. MEPHISTO FORMAL THEORIES

The Mephisto Semantic Framework ([3]) was devised as a canonical symbolic language for level 2 fusion. The framework presents a number of concepts across the five level hierarchy: Metaphysical; Environmental; Functional; Cognitive; and Social. The Metaphysical Level is founded on a process metaphysics through the introduction of concepts like identity, existence, time and space. The Environmental Level then classifies environmental aspects of the metaphysical world, including concepts like air, land and water. The Functional Level subsequently defines functional roles performed by elements of the metaphysical world. This allows for the introduction of concepts like movement, sensation, emission, strike, information, transformation, interpretation and attachment. The Cognitive Level extends the functional roles to cognitive roles, which facilitates the introduction of concepts like memory, belief, volition and attention. The Social Level follows with social concepts based around agreement and conflict. This engenders concepts of alliance, command, ownership and the like. Collectively the Mephisto concepts provide a rich framework for expressing situations in the world and conducting level 2 fusion.

Formal theories expressed in a formal language associated with a formal logic facilitate a semantic formulation of Mephisto concepts like those above. Section 3.3 in [4] discusses: model theoretic interpretations; Tarskian truth; Davidsonian meaning; formal theories; meaning; and intensional-sonian meaning; formal theories; meaning; and intensionality; to explain how symbols within a machine can be both meaningful to us and ensure that the machine respects those meanings when conducting inferences with those symbols. In essence the concepts are specified through a formal theory \( M \subseteq L \), being a set of sentences from a formal language \( L \). For example, the meaning of the spatial concept \( \text{connects} \) can be stated through the following formal theory \( M \) ([5]) from a first-order language \( L \) (e.g. [6]):

\[
\forall x(\text{exists}(x) \Rightarrow \text{connects}(x,x)). \\
\forall x \forall y(\text{connects}(x,y) \Rightarrow \text{connects}(y,x)). \\
\forall x(\neg \text{universal}(x) \Rightarrow \text{connects}(x,\neg x)). \\
\forall x \forall y \forall z((\text{exists}(x) \land \text{exists}(y) \land \text{exists}(z)) \Rightarrow \\
(\text{connects}(x,y+z) \equiv (\text{connects}(x,y) \lor \text{connects}(x,z)))). \\
\forall x(\text{exists}(x) \Rightarrow \exists y(\neg \text{connects}(x,y))).
\]

By defining rules for truth propagation in formal language \( L \), we acquire a first order logic \( \langle L, \vdash \rangle \) with truth preserving inference relation \( \vdash \subseteq P(L) \times L \), where \( P(x) = \{ u \in L \} \) for any set \( x \). \( \vdash \) is traditionally written in infix notation, and so for any set of sentences \( S \subseteq L \), \( S \vdash \sigma \) holds for any sentence \( \sigma \in L \) if and only if \( \sigma \) is true whenever all sentences in \( S \) are true. If we insist that \( M \) is always true, i.e. we take the formal sentences defining \( \text{connects} \) as axioms, then we effectively infer a consequence \( \sigma \) from any domain knowledge \( D \) if and only if \( (D \cup M) \vdash \sigma \). \( M \) then becomes a theory of meaning for \( \text{connects} \) by imposing truth conditions for \( \text{connects} \) that must always be preserved in any truth preserving inference. Any inference involving \( \text{connects} \) is always constrained by the conditions of formal theory \( M \).

Through the use of formal theories like \( M \) with associated logics \( \langle L, \vdash \rangle \), we are able to formally specify to humans what concept symbols like \( \text{connects} \) mean. But for a machine to respect those meanings when conducting inferences with those symbols, it is necessary to devise a logic \( \langle L, \vdash' \rangle \) in which the deductive inference relation \( \vdash' \) is specified entirely through syntactic manipulation rules. A logic \( \langle L, \vdash' \rangle \) is sound if for every \( S \subseteq L \) and \( \sigma \in L \), \( S \vdash' \sigma \) ensures \( S \vdash \sigma \). Sound logics guarantee that the symbol manipulation rules of inference are truth preserving. A logic \( \langle L, \vdash' \rangle \) is complete if for every \( S \subseteq L \) and \( \sigma \in L \), \( S \vdash \sigma \) ensures \( S \vdash' \sigma \). Complete logics guarantee that the symbol manipulation rules of inference recover all truth preserving inferences.

In 1930, Gödel proved soundness and completeness for first order logics ([7]). Thus the semantic consequences under \( \langle L, \vdash \rangle \) of any first order formal theory of meaning \( M \subseteq L \) are completely reducible by a set of syntactic manipulation rules \( \vdash' \) for \( L \).

For a machine to respect first order logic semantics, the remaining task is to program the syntactic manipulation rules \( \vdash' \) for \( L \). This effectively defines a logic \( \langle L', \vdash' \rangle \) in which \( L' \) is a programming implementation of \( L \) and \( \vdash' \) is a programming implementation of \( \vdash' \). Programs \( \vdash' \) that compute in accordance with a deductive inference relation \( \vdash' \) are historically termed theorem provers ([8]). Theorem proving usually relies on converting formulae expressed in \( L' \) into a clausal normal form and then applying \( \vdash' \) to the clause normal form expressions. The authors have developed their own Prolog implementation of \( \langle L', \vdash' \rangle \) to specify formal theories of meaning for Mephisto concepts. The following shows how the formal theory \( M \) for \text{connects} \) is expressed as a formal theory \( M' \) in the language of \( L' \) for machine computation.

\[
\text{all}(X, (\exists Y (\text{exists}(X) \Rightarrow \text{connects}(X,Y)))). \\
\text{all}(X, \text{all}(Y, (\text{connects}(X,Y) \Rightarrow \text{connects}(Y,X)))). \\
\text{all}(X, (\neg \text{universal}(X) \Rightarrow \text{connects}(X,\neg X))). \\
\text{all}(X, \text{all}(Y, \text{all}(Z, ((\text{exists}(X) \& \text{exists}(Y) \& \text{exists}(Z)) \Rightarrow \\
(\text{connects}(X,Y+Z) \equiv (\text{connects}(X,Y) \lor \text{connects}(X,Z))))). \\
\text{all}(X, (\text{exists}(X) \Rightarrow \exists Y, (\neg \text{connects}(X,Y))))).
\]
The authors' implementation includes two noteworthy aspects:

- a novel formula renaming technique to generate structure preserved and constructive Prolog clausal normal forms from first order formulae (section III); and
- a straightforward Skolem function eliminating technique that avoids infinite loops caused by Skolem functions (section IV).

Both techniques are provably truth preserving. These techniques have been implemented into a clausal normal form converter. When applied to input $M'$, the converter generates the following Prolog clausal normal form code.

| connects(A,A) :- exists(A). |
| connects(A,B) :- connects(B,A). |
| connects(A,B+C) :- connects(A,B), exists(C), exists(B), exists(A). |
| connects(A,B+C) :- connects(A,C), exists(C), exists(B), exists(A). |
| connects(A,B) :- connects(A,C), connects(B,A), exists(C), exists(B), exists(A). |
| universal(A) :- connects(¬(A),A). |
| or_clause([connects(A,B),connects(A,C)]) :- connects(A,B+C), exists(B), exists(C), exists(A). |
| connects(A,B) :- exists(A). |
| atom(A), skl13282321560(A)=B. |

Listing 1. The converter generated Prolog rules for $M'$.

The authors have also implemented a sound and complete meta-interpreter $\triangleright$ to facilitate inference with converted code (section V). The experimental results (section VI) show that the converter is not only able to generate structure-preserved normal clauses, but also avoids an exponential increase in the number of clauses.

III. FORMULA RENAMING

This section presents the formula renaming technique. The idea of formula renaming is to replace a subformula by introducing a proposition with a new predicate symbol. In [9], the structure-preserved approach suggests renaming all subformulae up to literals$^1$. This approach avoids an exponential increase in size, but it performs some unnecessary renaming. In contrast, the optimized approach in [10] and [11] only replaces a subformula if the replacement can decrease the number of clauses generated. While the optimized approach aims to generate a small clausal normal form for efficiency in theorem proving, the goal of generating Prolog clauses from first order formulae is to provide constructive solutions for queries. The structure-preserving and optimized approaches for formula renaming do not serve this purpose so well.

To illustrate, the following formula is a familiar definition (expressed as equivalence) of a vegetarian pizza from the semantic web ontology community:

$$\forall x (vp(x) \Leftrightarrow (p(x) \land \forall y (h(x, y) \Rightarrow vt(y))))$$

(1)

This definition asserts that "some thing is a vegetarian pizza if and only if it is a pizza and if it has a topping then it must be a vegetarian topping". If we consider only one direction of the definition $\forall x( p(x) \land \forall y (h(x, y) \Rightarrow vt(y)) \Rightarrow vp(x))$ then, the standard normalization procedure generates:

$$(\neg p(x) \lor h(x, skc(x)) \lor vt(p(x)) \lor (p(x) \land \neg vt(skc(x)) \lor vt(p(x))))$$

which can be equivalently expressed as the two implications:

$$p(x) \land vt(skc(x)) \Rightarrow vp(x).$$

$$p(x) \Rightarrow (p(x) \lor h(x, skc(x))).$$

where $skc(x)$ is a Skolem function$^2$ generated for eliminating the existential quantifier during transformation to clausal normal form. These generated clauses offer little advantage to a Prolog system when users query the generated program, however, because the original implication structure $\forall y (h(x, y) \Rightarrow vt(y))$ becomes lost during the normalization process. With the optimized approach in [10] and [11], a new predicate $R$ and a clause $\forall x \forall y ((h(x, y) \Rightarrow vt(y)) \Rightarrow R(x, y))$ are introduced to replace the implication subformula $\forall y (h(x, y) \Rightarrow vt(y))$, so that the structure of the original formula is preserved: $(p(x) \land R(x, y)) \Rightarrow vp(x)$. However, normalizing the newly introduced clause $\forall x \forall y ((h(x, y) \Rightarrow vt(y)) \Rightarrow R(x, y))$ forfeits the implication structure $\neg h(x, y) \Rightarrow R(x, y)$ and $vt(y) \Rightarrow R(x, y)$. These Prolog clauses do not give a sound answer to a query of the original definition. Thus, the optimized renaming approach does not adequately solve the structure preserving problem when it is applied to generate Prolog clauses.

This paper proposes a new formula renaming technique to address this problem.

Definition 1: (Formula renaming) Let $\varphi$ be a formula and $\phi$ be the subformula of $\varphi$ to be renamed and be in the form of $\neg(\alpha \Rightarrow \beta)$ or $\neg(\neg \alpha \lor \beta)$ or $\alpha \land \neg \beta$ or $\neg \exists x(\neg \alpha \lor \beta)$ or $\neg \exists x(\neg \alpha \land \beta)$ where $\alpha$, $\beta$ are formulae. Let $R$ be a new predicate symbol for $\varphi$ with arity $n$. The formula renaming strategy is then:

i) To replace $\phi$ in $\varphi$ with $\neg R(x_1, ..., x_n)$ where $x_1, ..., x_n$ are free variables within scope; and

ii) To separately add to the system the formula

$$\forall x_1, ..., x_n ((\neg \alpha \land (\alpha \land \beta)) \Rightarrow R(x_1, ..., x_n))$$

which in turn is normalized into $def^+ \land def^-$ such that:

$$def^+ \equiv (\neg p_1 \lor R(x_1, ..., x_n)) \land \cdots \land (\neg p_k \lor R(x_1, ..., x_n))$$

$$def^- \equiv (\neg \gamma_1 \lor R(x_1, ..., x_n)) \land \cdots \land (\neg \gamma_k \lor R(x_1, ..., x_n))$$

$^1$An atomic formula is a formula of the form $\forall x_1(\cdots \forall x_n(\alpha))$ where $\alpha$ is an $n$-ary relation symbol and each $x_i$ is a term referring to an object. A literal is an atomic formula or the negation of an atomic formula.

$^2$As part of the standard clausal normal form conversion, Skolem functions replace existential quantifiers with functions over variables within scope. For example, $q(x, skc1(x))$ is the standard clausal normal form of $\exists y q(x, y)$, while $q(x, skc2(x))$ is the clausal normal form of $\forall x \exists y q(x, y)$. The Skolem functions in effect simplify the expressions by functionally naming the existential element.
where $\alpha \land \beta \equiv \rho_1 \lor \cdots \lor \rho_j$ and $\neg \alpha \equiv \sigma_1 \lor \cdots \lor \sigma_k$ are in disjunctive normal form.

Significantly, the normalization of the newly added clause in ii) avoids structural damage to the original formula:

$$\forall x_1\ldots x_n(\neg \alpha \land (\alpha \lor \beta)) \Rightarrow R(x_1,\ldots,x_n)$$

$$\equiv \forall x_1\ldots x_n[((\alpha \lor \beta) \lor (\neg \alpha) \Rightarrow R(x_1,\ldots,x_n)]$$

$$\equiv \forall x_1\ldots x_n(\neg (\alpha \land \beta) \lor \neg \alpha \lor R(x_1,\ldots,x_n))$$

As $\alpha$ and $\beta$ in the subformula $\phi$ can be any formula, they may require further normalization.

To illustrate, the vegetarian pizza example uses the equivalence in Eq (1), which, when converted to normal form, initially produces the two implications:

(a) $\forall x (vp(x) \Rightarrow (p(x) \lor y(h(x,y) \Rightarrow vt(y))))$; and

(b) $\forall x ((p(x) \lor y(h(x,y) \Rightarrow vt(y))) \Rightarrow vp(x))$.

Thus, (a) can be expressed in Prolog by the two rules:

$$p(X) : = \neg vp(X),$$

$$vt(Y) : = \neg vp(X), h(X,Y).$$

Converting (b) into its normal form produces:

$$\forall x(\neg p(x) \lor \exists y(\neg(h(x,y) \lor vt(y)) \lor vp(x)).$$

along the way, with $\exists y(\neg(h(x,y) \lor vt(y))$ as a subformula of the form $\exists y(\alpha \Rightarrow \beta)$ since $\alpha \Rightarrow \beta =_{def} \neg \alpha \lor \beta$. Consequently, in accordance with part i) of the renaming strategy in Definition 1, a replacement $\neg R(x,y)$ is substituted for $\neg(h(x,y) \lor vt(y))$ in the formula to yield

$$\forall x(\neg p(x) \lor \exists y(\neg R(x,y)) \lor vp(x)).$$

which can be re-expressed as

$$\forall x ((p(x) \land \forall y(R(x,y))) \Rightarrow vp(x).$$

and so can be represented in Prolog by the rule:

$$vp(X) : = \neg p(X), R(X,Y).$$

In accordance with ii), the formula

$$\forall xy((\neg h(x,y) \lor (h(x,y) \land vt(y)) \Rightarrow R(x,y))$$

is independently added to the system, which can be equivalently expressed by

$$\forall xy(\neg h(x,y) \Rightarrow R(x,y)) \land \forall xy((h(x,y) \land vt(y) \Rightarrow R(x,y)).$$

and so can be represented in Prolog by the rules

$$R(X,Y) : = \neg h(X,Y),$$

and $R(X,Y) : = h(X,Y), vt(Y).$

Therefore the collective effect of (a) and (b) results in the Prolog rules shown in Listing 2 to compute Eq (1), where the suggested generated predicate symbol $R$ is in fact $pred13280667870$.

---

**Theorem 1** Let $\phi$ be a subformula in $\varphi$ and by applying Definition 1, the rename of $\varphi$, $rename(\varphi, \phi)$ be

$$\varphi[\phi/\neg R(x_1,\ldots,x_n)] \land \forall x_1\ldots x_n(\phi \Rightarrow R(x_1,\ldots,x_n))$$

Then, $rename(\varphi, \phi)$ is satisfiable if and only if $\varphi$ is satisfiable.

**Proof.** We need to show that $\varphi$ is a logical consequence of $rename(\varphi, \phi)$ and any model of $\varphi$ can be expanded to a model of $rename(\varphi, \phi)$.

For the $\Rightarrow$ direction, assume $M \models rename(\varphi, \phi)$. In the formula of $rename(\varphi, \phi)$, if $R(x_1,\ldots,x_n)$ holds, then $\phi$ must hold since $\phi$ is replaced by $\neg R(x_1,\ldots,x_n)$ by definition; on the other hand, if $\neg R(x_1,\ldots,x_n)$ holds, then $\neg \phi$ must also hold. Thus, $\varphi$ is satisfiable if $rename(\varphi, \phi)$ is satisfiable.

Now for the $\Leftarrow$ direction, assume $M \models \varphi$. Then we have $M = (I,D)$, where $I$ is the interpretation over the domain $D$. Since $R(x_1,\ldots,x_n)$ is new to $\varphi$, we can construct a model $M' = (I',D)$ to include $M$ and an appropriate interpretation for $R(x_1,\ldots,x_n)$, e.g. $(a_1,\ldots,a_n) \in R(x_1,\ldots,x_n)$. $\square$

Now we present the renaming algorithm. Given a formula $\varphi$ after eliminated implications and equivalences, rename all subformulas $\phi$ occurrences in the form of $\neg(\neg \alpha \Rightarrow \beta)$ or $\neg(\neg \alpha \lor \beta)$ or $\alpha \land \neg \beta$ or $\neg \forall x(\neg \alpha \lor \beta)$ or $\neg \exists x(\neg \alpha \lor \beta)$, and return a new formula, $rename(\varphi, \phi)$.

The algorithm implements the outside-inside strategy to traverse the formula tree. It terminates because it traverses the formula tree from the root to the leaves. This algorithm is applied after the implication elimination step and before the negation normal form step in the standard clausal normal form conversion. Since the structure damage to the original formula and the exponential increase in the size of clauses generated may occur only during the distribution of conjunctions over the disjunctions, the candidate renaming subformula can be identified more easily.

#### IV. Skolem Function Elimination

One of the problems with converting formulae to a normal form is that the Skolem functions introduced for eliminating existential quantifiers can cause infinite loops in the resulting Prolog program. For example, a formula like:

$$\forall x(w(x) \Rightarrow \exists y((f(x,y) \lor m(x,y)) \land w(y)))$$

will produce the subformula $w(x) \Rightarrow w(skc(x))$, where $skc(x)$ is a Skolem function. As a Prolog rule this becomes $w(skc(X)) : = \neg w(X)$, which potentially causes infinite loops when executed.
Algorithm 1: rename($\varphi$)

Input: $\varphi$

Output: rename($\varphi$)

begin

if $\varphi$ is in the form of $\alpha \land \neg \beta$ or $\neg (\alpha \lor \beta)$ or
\neg(\forall x(\neg x \lor x)) \lor \neg(\exists x(\neg x \land \beta)) then
 return $R(x_1, \ldots, x_k) \land \text{def}^+ \land \text{def}^-$

if $\varphi$ is in the form of $\forall x\delta$ then
 return $\forall x(\text{rename}(\delta))$

if $\varphi$ is in the form of $\exists x\delta$ then
 return $\exists x(\text{rename}(\delta))$

if $\varphi$ is in the form of $\delta_1 \land \delta_2$ then
 return $\text{rename}(\delta_1) \land \text{rename}(\delta_2)$

if $\varphi$ is in the form of $\delta_1 \lor \delta_2$ then
 return $\text{rename}(\delta_1) \lor \text{rename}(\delta_2)$

if $\varphi$ is in the form of $\neg (\alpha \land \beta)$ then
 return $\text{rename}(\neg) \lor \text{rename}(\beta)$

if $\varphi$ is in the form of $\neg (\alpha \lor \beta)$ then
 return $\text{rename}(\neg) \lor \text{rename}(\beta)$

if $\varphi$ is in the form of $\neg \alpha$ then
 return $\text{rename}(\neg) \lor \text{rename}(\alpha)$

if $\varphi$ is in the form of $\neg \delta$ then
 return $\neg \text{rename}(\varphi)$

else
 return $\varphi$

end

A number of optimized Skolemization techniques ([10], [11], [12], [13]) have been proposed to improve standard Skolemization, leading to a reduction of search space and proof length, but they do not address the infinite loop issue in the context of logic programming. An approach to eliminate Skolem functions was presented in [14], where the procedure of clausal normal form transformation is modified to eliminate Skolem functions, with relations introduced instead of Skolem functions in the Skolemization step of the procedure. This modified transformation has technical complications with its implementation, however.

In this section, we present a straightforward way of eliminating Skolem functions in the context of logic programming. The basic idea is that instead of modifying the transformation procedures, we replace Skolem functions by introducing variables and relations after the full transformation procedure is completed, where other procedures such as simplification, formula renaming, negation normal form, anti-prenexing, optimized Skolemization and clause factoring may be included.

Definition 2 (Skolem function elimination) Let the clause normal form of a formula $\varphi$ be $C_{nf} = \{c_1, \cdots, c_n\}$ and $c_i$ be in the disjunctive normal form $d_1 \lor \cdots \lor d_m$ where $d_j$ is a literal. Let $S_{k_f}$ be the set of Skolem functions in $C_{nf}$ and $V_{ars}$ be the set of variables new to $C_{nf}$ where $f$ is the mapping $f : S_{k_f} \mapsto V_{ars}$. Then, Skolem function elimination of $C_{nf}$ comprises two things:

- Substitute every Skolem function, $skf$ in $d_j$ by a variable, $v = f(skf)$ for every clause $c_i$ in $C_{nf}$, where $v \in V_i$ and $skf \in S_{k_f}$ in $c_i$.
- For each of the Skolem function substitutions $skf/v$, create an equal relation $\delta$ to be $skf = v$ and add the negation of the equal relation $\neg \delta$ as a disjunction to the clause $c_i$; $d_1 \lor \cdots \lor d_m[skf/V_{ars}] \lor \neg_1 \lor \cdots \lor \neg_{l-1}$, where $l \geq 0$ and $r_i \in \{skf = v_i | skf \in S_{k_f}, v_i \in V_i\}$.

If a skolem function $skf$ contains only a single variable $x_i$, a literal $\neg \text{atom}(x_i)$ is added to $c_i$ where $\text{atom}(x_i)$ is a built-in predicate in a Prolog system and $x_i$ is the variable argument of the Skolem function $skf_i$.

Notice that the addition of the term $\neg \text{atom}(x_i)$ is to avoid infinite loops in a Prolog system. Basically, the purpose of Skolem function elimination is to substitute all Skolem functions by new variables and introduce equation relations into the clauses.

To illustrate this, consider Eq (2) as the example formula.

Application of the clausal normal form transformation to Eq (2) produces the logically equivalent conjunctive normal form\(^4\) representation

\[
(f(x, skc(x)) \lor m(x, skc(x)) \lor \neg(w(x))) \land (w(skc(y)) \lor \neg w(y)).
\]

Applying the aforementioned Skolem function elimination, results in the formula

\[
(f(x, z) \lor m(x, z) \lor \neg w(x) \lor (skc(x) = z) \lor \neg \text{atom}(x)) \\
\land (w(z) \lor \neg w(x) \lor (skc(x) = z) \lor \neg \text{atom}(x))
\]

which can be expressed in Prolog-like notation by the two rules:

\[
\begin{align*}
&f(X, Z) \lor m(X, Z) : \neg w(X), \text{atom}(X), skc(X) = Z. \\
&w(Z) : \neg w(X), \text{atom}(X), skc(X) = Z.
\end{align*}
\]

The actual output from the converter is shown in Listing 3.

Listing 3. The generated Prolog rules for Eq (2) from the converter.


\[
\text{m(A,B)} := "f(A,B), w(A), \text{atom}(A), sk13280673450(A)=B.}
\]

\[
f(A,B) := "m(A,B), w(A), \text{atom}(A), sk13280673450(A)=B.}
\]

\[
w(A) := w(B), \text{atom}(B), sk13280673450(B)=A.}
\]

\[
or_{\text{clause}} ([m(A,B), f(A,B)]) :=
\]

\[
w(A), \text{atom}(A), sk13280673450(A)=B.}
\]

Theorem 2 establishes that Skolem function elimination is truth preserving.

Theorem 2 Let $c_i = d_1 \lor \cdots \lor d_n$ be a normal clause in the conjunctive normal form $C_{nf}$ of a formula $\varphi$, and $nc_i$ be the new clause of $c_i$ generated by applying the Skolem function elimination as described in Definition 2. Then, $c_i$ is satisfiable if and only if $nc_i$ is satisfiable.

Proof. This follows straightforwardly by applying the substitution axiom of the equality theory [15]:

\[
q(x_1, \cdots, x_m) \leftrightarrow q(y_1, \cdots, y_m) \cup \{x_i = y_i | 1 \leq i \leq m\}
\]

\(^4\)The conjunctive normal form of a formula is logically equivalent to the formula and consists of a conjunction of disjunctions of literals.
V. META-INTERPRETER

This section briefly describes the meta-interpreter which has been designed for interpreting the normal clauses transformed by the converter presented in the previous sections.

As an extension of Horn clauses, a clause converted from a first-order logic formula through the aforementioned process may contain a disjunctive head and/or one or more strong negations in the body, and thus have the form $P_1 \lor \cdots \lor P_m \leftarrow Q_1 \land \cdots \land Q_n$, which the authors convert into the following Prolog clause:

\[
\text{or_clause}([P_1, \ldots, P_m]) : - Q_1, \ldots, Q_n, \text{and its contrapositive clauses}
\[

P_1 : - Q_1, \ldots, Q_n, \neg P_2, \ldots, \neg P_m.
\]

\[
\vdots
\]

\[
P_m : - Q_1, \ldots, Q_n, \neg P_1, \ldots, \neg P_{m-1}.
\]

where \text{or_clause} is a second-order predicate for handling disjunctive heads in a Prolog program. The following Prolog rule is defined to make weak negation (negation as failure, represented by $\neg$ in SICStus Prolog) serve as strong negation $\neg P : - \neg P$. The rule extends the Herbrand interpretation base to be \text{Atom}/\text{N_atom}, where \text{Atom} is the set of ground atom and \text{N_atom} is the set of explicit negation atoms.

Most Prolog engines are designed for Horn clauses and are sound and complete for Horn clauses only, with infinite loops being hard to detect. It is therefore appropriate to extend a Prolog engine to a meta-interpreter with features similar to that of the Prolog Technology Theorem Prover (PTTP) ([16]). These features include model elimination by ancestor resolution; an iterative deepening search strategy; simple loop detecting; unification with occurs checking; support for disjunctive queries and disjunctive head queries; and display of the proof tree. The model elimination and iterative deepening search strategy features ensure that the meta-interpreter is sound and complete. Listing 4 below is a fragment of the authors’ meta-interpreter source code implemented in SICStus.

The meta-interpreter implements the features mentioned above. The \text{demo} predicate contains five arguments: the first is the goal $G$ to query; the second is an ancestor list $A$; the third is a proof tree $\text{tree}(G,T)$ to display; the fourth is the depth $D$ to search and the fifth is the depth limitation $L$ for the iterative deepening search. As can be seen, the meta-interpreter also implements simple loop checking and unification with occurs checking. The \text{demo} predicate supports two types of disjunctive queries: the first is to search if one of the disjunctions is true; and the second is to construct an \text{or_clause}([G_1, \ldots, G_n]) to search to determine whether the \text{or_clause} is true by calling the \text{demoDisj} predicate, which supports disjunctive heads of the form \text{or_clause}([P_1, \ldots, P_m]) : - Q_1, \ldots, Q_n.

The \text{demoDisj} predicate implements the addition and disjunctive syllogism inference rules by calling the predicate \text{demo}((Gs), A, T, D, L). For example, given a program:

\[
\{\text{or_clause}(a, b, c, d) : - e, \neg e, e, e\}
\]

we can prove $\alpha \lor \beta \lor d$ to be true by applying the addition rule $\{\alpha \lor \beta\}$; and prove $\alpha \lor \beta \lor d$ to be true by applying the disjunctive syllogism rule $\{\alpha \lor \beta, \neg \beta \lor \alpha\}$. Listing 5 below shows a fragment of the implementation of the predicate \text{or}(Gs, A, T, D, L) in SICStus Prolog for the two inference rules.

\[
\text{Listing 4. A fragment of the meta-interpreter source code in SICStus}
\]

\[
\text{Listing 5. A fragment of implementation of the addition and disjunctive syllogism inference rules in SICStus.}
\]

VI. EXPERIMENTAL RESULTS

A converter based on the techniques described in the previous sections, in addition to the meta-interpreter, has been implemented in SICStus Prolog. The 75 Pelletier example formulae for testing automated theorem provers in [17] and 4 other challenge problems from [18] and [19], have been used for this experiment.

All the 75 example formulae were converted successfully and the results have been compared with the optimized approaches in [10] and [11]. Of the 75 formulae, the optimized approach renamed 17 and our new approach renamed 21. 11 formulae are commonly renamed. The new approach didn’t rename the example formula 29 and 53 which resulted in larger numbers of clauses compared with the renamed outputs from the optimized approach (30 to 12 and 130 to 18 respectively). Table 1 shows the results from the optimized approach, the standard procedure, and the new renaming approach for the 4 challenge problems. It can be seen that our new approach can also dramatically reduce the number of clauses in addition to avoiding damage to the structure of the original formula.
VII. CONCLUSIONS

Under the STDF model, level 2 fusion forms assessments of situations in the world, with each situation consisting of a sequence of states of affairs, and with each state of affairs represented by a set of propositions. To facilitate fusion, a common canonical language is required to represent these propositions. In the first instance the language must canonically represent a diversity of information deriving from level 1 signal, textual and imagery processing. In this way the canonical language comes to uniformly represent information from, for example: radar returns; overhead imagery returns; and human intelligence reports. To facilitate fusion with the canonically represented information, however, additional inference is required beyond perceptual information, drawing upon both background knowledge and an understanding of the meaning of the concepts used. The choice of concepts for this purpose is fundamental. Mephisto has been developed with this task in mind.

Having selected appropriate concepts, the challenge is to clearly formulate the meaning of those concepts and to determine how a machine might appropriately operate with those concepts. Through the use of formal theories with an accompanying formal logic, it is possible to rigorously specify the meanings of the Mephisto concepts. Moreover, when these meanings are expressed through first order theories, the semantic consequences of those concepts are provably reducible to syntactic symbol manipulations rules, which can in turn be programmed within a machine. The corollary is that the machine should then be able to meaningfully operate with formalized Mephisto concepts like time, air, interpretation, intent and authority.

To achieve this, the syntactic symbol manipulations rules must be efficiently implemented within the machine. The authors have implemented an approach based on logic programming, while including a couple of noteworthy differences from standard approaches. A formula renaming technique has been applied to generate constructive Prolog clauses from standard approaches. A formula renaming technique has been introduced clause α first order logic formula. The key ideas of this technique are been applied to generate constructive Prolog clauses from standard approaches. A formula renaming technique has been implemented an approach based on logic program work", DSTO Technical Report TR-2162, Department of Defence, 2008.


