***Confidential to Coherent Knowledge Systems***

**Specification of Omniformity Transform with comments about its Ergo implementation**

Benjamin Grosof and Michael Kifer

Resent log entries::

V32 of 3/1/2016 by MK: additional comments about transforms that don’t seem to have use cases and thus were not implemented

V31 of 10/17/2015 by MK: do not push \naf past AND if the sides have common free variables

v30 of 9/30/2015 by MK: fixed omission of pushing exists through disjunction in the body transform

Old log entries:

v10 of 8/8/2013 by BG

v11 of ~12/13/2013 by MK

v12 of 12/18/2013 by BG: wove in misc. material from -more-details doc, plus some misc. polishing edits

v13 of 12/18/2013 by BG: small tweaks

v14 of 12/18/2013 by BG: major reorganization to better treat the composition of head sub-transform with body sub-transform, and associated generation of refutable vs. irrefutable auxiliary info

v15 of 12/28/13 by BG: finished smoothing the v14 reorganization, then did a major refactoring and revision to treat the revised LT which produces NF1 rather than NF2. Change tracking is still with respect to v11.

v16 and v17 of 1/2/14 by MK: lots of comments and edits

v18 and v19 of 1/9/14 by BG: Removed early/late material on other aspects of overall Rulelog transform such as AT-defeasibility, Hilog, and restraint. Renumbered the sections accordingly. Addressed technical issue on neutrality aspect of TNF, in comment BG28 or so. Some other minor polishing edits. Still retains most of the tracked changes proposed by MK in v17.

v20 of 1/9/14 by BG: Accepted changes made during v18 and v19. Other minor polishing edits oriented mainly towards version control. Still retains most of the tracked changes proposed by MK in v17. Accepted other changes made during v18-v19. Note some changes proposed by MK in v17 were rejected.

v21 of 1/9/14 by BG, and v22 of 1/12/14 and v23 of 1/13/14: by BG Accepted changes made during v20. Accepted most changes (and comments) by Michael, but rejected some of those, including a few followup corrections related to the neg-not-thru-naf issue. Many major and minor polishing edits oriented towards clarity and brevity, per Michael requests. Result is that the document is oriented towards implementation, omits NF2 as the target form and macro introduction and associated transform variants, and has nearly a minimum of terminology and nearly a minimum of repetition. Neutrality comment is now BG7. OC (combination step) replaced by discussion in OH-C of how OB and body consolidation are applied after directionalization. Added new material on "trick" for keeping directionalization blowup worst-case linear.

V24: by MK Accepted most of the changes by BG. Many stylistic changes, clarifications, improvements, suggestions, comments. The biggest changes: completely deleted the first description of L-T: was redundant and incorrect. See the comments. Completely deleted the mention of sensors in the head. This is not possible and should not be allowed. We talked about it. Further discussion of pushing quantifiers through *and* and *or*. Need to resolve the issue of the comments BG22/MK23.

V25: by MK Significant: more on how and to drop ∃-quantifiers in L-T, p.7. When to break *forall* and also on pushing *naf* past *exists*, p.7. Also, about the unsoundness of Skolemization, p. 11.

V26 by MK on 1/16/14: ft1/2 changed to NF2/1. Clarified the treatment of forall in OB. New remarks on pushing quantifiers in OH-TNF. Remarks on built-ins/sensors in the rule head.

v27 by BG on 1/16/14: Misc. polishing. Refined the head sensors discussion (6.B.). Added more discussion on blowup complexity (6.F.). (No changes of substance to the transform itself.)

v28 by MK: Minor changes. Added comments.

v29 by BG on 1/18/14: Minor changes, mainly oriented to future.

To consider using in future versions: much of the material that Michael

deleted in v17 (still visible in tracked changes in v19-20 and largely v21). Notably:

- body consolidation section

- OC and its section

- stuff on bookkeeping meta-data

- stuff on NF2, and macros incl. macro introduction

- stuff on intermediate-body introduction

- stuff on cranial etc. rules

- stuff on subgoal reordering heuristic optimization

- additional location of discussion of unsafe head variable warnings in general

- additional locations of discussion of variable renaming

**0. Misc. Notes**

Notation:

* Below, H, B, G, J, etc. stand for formulas.
* "--->" stands for "transforms into".

Terminology: A formula is said to be "*neutral*" with regard to - a variable if that variable does not appear within the formula. (This is used in some steps when converting to TNF.)

**Terminology**: A formula moved to the body (arising) from the head via directionalization is called "*cranial.*"

A formula (whether head or body) can be viewed as an *expression tree*.

* In a *basic* expression tree, each single-variable quantifier is a node and each connective is a node. In addition, each atom is a node that, moreover, is a leaf. The outermost quantifier or connective is the root of the tree, a.k.a. the *top* of the tree. Some types of nodes (forall, exist, neg, naf) have a single child. Other types of nodes (And, Or; also: strong implication, strong equivalence, weak implication, weak equivalence) have two children.
* In a general-form ("*complex"*) expression tree, the expression tree abstraction is further generalized to permit a single node to be a multiple-variable quantifier (e.g., "exist ?x,?y,?z") or a n-ary conjunction/disjunction (where n>2, e.g., "p and q and r"). An n-ary conjunction/disjunction has n children.

The overall omniformity feature transform is called: OT.

**1. General**

A Rulelog rule has the form:

@!{c} @{p} H :- B .

Here, c is a constant, p is a constant, H is a head formula, and B is a body formula. H :- B is called the "main" part/expression of the rule. The descriptors @@{strict} and @@{defeasible} may additionally indicate that the rule is strict or defeasible. Alternative supported syntax for the annotation part is: @!{c[tag->p, defeasible]}. Additional, *user-defined* attribute-value pairs can be supplied in the frame attached to the @! descriptor. For instance, @!{c[tag->p, strict, foo->bar]}. The descriptor syntax has a number of additional features that are not relevant to the present discussion, but can be found in the Ergo manual.

The tag value p can be generalized to be any hilog term p(t1,...,tn), where p is still a constant, but whose free variables are a sub-tuple of the free variables appearing in the main part of the rule. Ditto, rule id's can be any hilog term.

A head formula has the form of hilog/F-logic first order logic. A body formula is similar, but may mention the following additional connectives: naf, weak implication ~~>, and weak equivalence <~~>, as well as builtins and aggregate operators. Ergo furthermore permits update operators in the body, but actions cannot be defeated. The update operators (cf. Transaction Logic) are not part of the current Rulelog spec per se, but are an interesting and useful extension.

Neg outside of naf is essentially disallowed: As usual in LP and its extensions: Essentially, ***neg is not permitted to appear outside the scope of naf***. If it does so appear, this is treated as an error in syntax of the rule. The reason this is stated as "essentially", rather than flatly (unconditionally), is that in some cases, however, one can relax this condition a bit because this pattern of appearance is inessential -- i.e., "remediable" by (allowed) transformations. The section on the generalized Lloyd-Topor transform (OB) gives more details. For example, the rule

q(?z) :- neg exist(?x)^(naf forall(?y)^(naf p(?x,?y,?z))). /\* neg appears outside of naf, inessentially \*/

transforms, via the body sub-transform, into the rule

q(?z) :- forall(?x)^ (forall^(y)^(neg p(?x,?y,?z))). /\* neg does not appear outside of naf \*/

The reason for this restriction is that we want neg F |= naf F, for any F. This is a widely accepted semantic property of neg/naf.

**2. Overview of OT**

The overall omniformity (omni) transform OT takes an *input* rule and outputs a set of rules, which are called output rules. OT is composed from two key sub-transforms OH and OB. OH and OB are each described in their own subsection(s) below.

OH: Head formula sub-transform. OH takes a rule as input, and transforms that rule's head formula resulting in a number of rules each having just plain literals (possibly *neg*-negated) in their heads. OH is an innovative transform; it is the most important aspect of the overall omni transform. OH introduces/"pushes" new formulas into the body, during directionalization of head disjunction. These formulas are combined with the body of the transformed rule.

OB: Body formula sub-transform. OB takes a rule as input, and transforms that rule's body formula using an extension of the Lloyd-Topor (LT) transform that generalizes LT to treat *neg*  as well as the common features provided by every Prolog engine, such as body-disjunction and naf\_exists (available in XSB).

The OH and OB transforms are independent of each other and can be applied in any order. However, the most straightforward way is to first apply OH, then for each rule resulting from OH combine its body (the "cranial body", a.k.a. "head-to-body residue") with the input rule’s (original) body, and then apply OB to that combined body.

**\*\*\*TODO**:

with respect to when in OT one avoids doing LTM (LTM transforms from NF1 to NF2):

Revisit some related matters for defeasibility and justification:

1. we probably need to modify the ATCO defeasibility argumentation theory: to use the more general notion of refutable body formula (rather than literal), and particularly to modify the part of ATCO2 that treats naf'd macro-predicate literals.

2. In the justification graph (JG), we can view various (non-literal) body (sub-)formulas as "virtual nodes" that correspond implicitly to macro literal formulas in LTM. Then, as a meta-default, we can hide those virtual nodes when presenting the JG.

**3. Preliminaries**

A. General Terminology

*strong literal*: atomic formula or *neg* of atomic formula.

*fact*: a rule that has an empty body and a strong literal as head.

B. Target Form: Definitions and Terminology

In the First Normal Form (NF1), the head is a single strong literal, but the body is a formula built up out of (strong) literals in which { *and, or, naf*, *exist* } (but not forall) can appear freely nested, except for the following restrictions (which are discussed in more detail below in the section on Lloyd-Topor):

1. *naf* can appear immediately outside of a strong literal or *exist* , but does not appear immediately outside of *naf,* *and,* or *or*  (i.e., *naf* is fully pushed through *and*, *or*, and *forall,* but it is not pushed through *neg* and *exist*).
2. *exist* does not appear anywhere except immediately after *naf*.

The rationale behind NF1 is that these rules can be evaluated directly by any Prolog system (augmented by a few auxiliary rules to define the *naf exist(...)* combination, as is done in XSB).

There is also the Second Normal Form, NF2, which differs from NF1 in that *or* and *exist* are completely eliminated. In NF2, macros (new predicates) are introduced. We do not consider NF2 in this document, however.

An "input" rule is the pre-transform rule.

- Such a rule may contain omniform heads.

An "output" rule == a rule that is in the output of OT, i.e., "post-transform"

For each input rule there is a set of output rules.

OB and OT each output NF1 rules, as does LT2.

**4. OB (Body formula sub-transform) -- Extension of Lloyd-Topor that Treats Neg**

OB-NF1 (or just OB), the body formula sub-transform, is the extension of the Lloyd-Topor (LT) transform to treat *neg* as well as to take into account the facilities provided by Prolog engines, like XSB, including the support for body-OR and naf\_exist. OB outputs rules in NF1.

Let the input rule RI have the form @{p} H :- B.

OB outputs a single NF1 rule: @{p} H :- B2. This rule has the same head and tag as the input rule, but its body B2 is "simpler" in in the sense that it does not have quantifiers and can be evaluated directly by a Prolog engine.

In OB, *neg* may appear in the body (and head) of the input rule and thus in the bodies (and heads) of the output rules as well, unlike in LT. In OB, strong implication and strong equivalence may also appear in the body (and head) of the input rule, unlike in LT. Strong implication and strong equivalence reduce to the *neg-or* combination. As in LT, weak implication can appear in OB in the input rule bodies.

OB modifies only the body formula of the input rule. The steps are as follows.

1. Eliminate strong and weak equivalence and strong and weak implication:
   1. Strong equivalence (A <==> B) is reduced to strong implication ((A ==> B) and (B ==> A)).
   2. Weak equivalence (F <~~> G) is reduced to weak implication ((F ~~> G) and (G ~~>F)).
   3. Strong implication (F ==> G) is reduced to disjunction ((neg F) or G).
   4. Weak implication (F ~~> G) is reduced to disjunction ((naf F) or G).
2. Next: Push neg and naf inward as much as possible, but: without pushing neg past naf, without pushing naf past neg, and without pushing naf past *exist.* "As much as possible" here means repeat doing steps (a)-(c) below until exhaustion. This pushing of *neg* and *naf* is part of the semantic definition of these connectives.
   1. naf is pushed inward (but not past *exist*), as follows:
      1. *naf and* -> *or naf*

Do the above push **only** if the conjuncts don’t have free variables in common. This is because *naf (P(?X)* **and** *Q(?X))*  would become *naf p(?X)* **or** *naf Q(?X)* and if ?X is unbound then it is interpreted as *naf exists(?X)^p(?X)* **or** *naf exists(?X)^Q(?X)* – a completely different meaning compared to *naf exists(?X)^(P(?X)* **and** *Q(?X))*  .

* + 1. *naf or* -> *and naf*
    2. *naf naf* ->∅(no connective, double-naf cancels out)
    3. *naf forall* -> *exist naf*
    4. *naf exist -> forall naf*
    5. *naf neg* -> no change, i.e., is left as is
  1. neg is pushed inward, as follows:
     1. *neg and* -> *or neg*
     2. *neg or* -> *and neg*
     3. *neg neg* ->∅(no connective, double-neg cancels out)
     4. *neg forall* -> *exist neg*
     5. *neg exist* -> *forall neg*
  2. *neg naf* -> no change, i.e., is left as *neg naf*.
  3. Push *exist*'s past the *or*'s when applicable. Also push them through 1-sided ANDs (ie, when only one conjunct has the existential vars). Do the same for forall: propagate through AND and 1-sided OR. We push foralls even though we later replace them with naf-exists-naf. Reason: nafs can’t be as easily propagated through AND. Test case: general\_tests/lt3.flr.
  4. For ***any*** remaining *forall* quantifier, eliminate itas described below and do some additional pushes:

1. Replace each remaining *forall* with *naf* *exists* *naf*. For instance:

*forall*(?vars)^(G(?vars)

becomes

naf\_*exist*(?vars)((*naf* G(?vars)).

1. Push *exist* inwards past the *or*, if applicable; push them also through 1-sided ANDs.
2. Push the inner naf as before, but this time ***not*** past *exists*

**Step (ii)** above is not being done as it requires an additional scan, and we do not have testable usecases where (ii) makes a difference (changing undefined results to true/false). Note: we do perform this step before (e).

In contrast, **step (d)** above *does* have a testable use case: general\_tests/lt3.flr. Without case d, tst1 and tst2 will give different results (rule r1/2 will give undefined answers while r2 will give true answers).

1. Next: If (in the result of (2)) *neg* appears outside of *naf*, report a syntax error because, as explained earlier, *neg* is not permitted to appear irremediably/essentially outside the scope of naf.
2. Next: If a rule body has an *exist* that is not in the scope of any *naf*,then drop that *exist* but rename the variables quantified by that *exist*. Merge the remaining *exist*'s with their containing *exist*'s. For instance:

foo, *naf* *exist*(?X)^(abc(?X), *exist*(?Y)^(foobar(?X,?Y)))

becomes

foo, *naf* *exist*(?X,?newvar)^(abc(?X), foobar(?X,?newvar))

**5. OH (Head formula sub-transform) -- Generate Unit-Head Cranial Rules**

A. Overall

OH, the head formula sub-transform, takes as input a rule RI. It then applies the following transforms, in sequence:

1. **OH-TNF**, the preparatory phase of OH, which converts the head formula to the Tight Normal Form (TNF).
2. **OH-C** generates cranial rules (whence the “C” part in OH-C) with simpler heads. In terms of the root node of the head's expression tree:
   * An existential is skolemized.
   * A disjunction is directionalized.
   * A conjunction is split.
   * A universal is dropped.
   * A literal is left unchanged.

Each output rule inherits the input rule's tag, and meta-data is added.

1. OH-C applies iteratively to the result of each previous iteration until no changes occur. This generation is performed by recursing on the rule head's expression tree. The recursion finishes when the rule heads are unit literals. At that point, the generated rules have only simple heads, i.e., they are atoms or *neg*-negated atoms.

B. OH-TNF: Convert to Tight Normal Form (TNF)

TNF addresses a subtlety that directionalization should be done before skolemization. TNF differs, in general, from Skolem Normal Form (SNF, which is used in (Hilog/F-Logic) FOL resolution theorem proving).

TNF has a pattern that can be summarized as (F?O?E?A?)\*(L) where F stands for forall, O stands for *or*, E stands for *exist*, A stands for *and*, L stands for a strong literal, \* stands for the usual Kleene star operator, and ? means that the preceding pattern (F,O,E, or A) can appear at most once.

In more detail:

1. Strong equivalence (G <==> J) is reduced to strong implication ((G ==> J) and (J ==> G)).

2. Strong implications (G ==> J) and (G<==J) are reduced to disjunction ((neg G) or J).

3. *Neg* is driven inward as follows:

1. *neg-and -> or-neg*: neg (G1 and ... and Gn) is replaced by (neg G1 or ... or neg Gn)
2. *neg-or -> and-neg*: neg (G1 or ... or Gn) is replaced by (neg G1 and ... and neg Gn)
3. *neg-exist -> forall-neg*: neg exist x. G[x] is replaced by forall x. neg G[x]
4. *neg-forall -> exist-neg*: neg forall x. G[x] is replaced by exist x. neg G[x]
5. *neg-neg ->* ∅ , i.e., the double-neg cancels out. For instance, neg neg G is replaced by G

4. Put into Tight Normal Form (TNF), by doing each of the following wherever possible:

a. push *exist* inward past *or* ;

b. push *forall* inward past *and* ;

c. push *forall* inward past *or*, **when the disjunct does not mention any of** that *forall's* quantified variable(s), i.e., when the disjunct is neutral; and

d. push *exist* inward past *and*,

**when the conjunct does not mention** any of that *exist's* quantified variable(s), i.e., when the conjunct is neutral.

Examples:

* 4a. push *exist* inward past *or* :

exist(?x)^(p(?x) or q(?x))

--->

exist(?x)^(p(?x)) or exist(?x)^(q(?x))

* 4b. push *forall* inward past *and* :

forall(?x)^(p(?x) and q(?x))

--->

forall(?x)^(p(?x)) and forall(?x)^(q(?x))

* 4c. push *forall* inward past *or* , when disjunct is neutral:

forall(?x)^(p or q(?x))

--->

p or forall(?x)^(q(?x))

* 4d. push *exist* inward past *and* , when conjunct is neutral:

exist(?x)^(p and q(?x))

--->

p and exist(?x)^(q(?x))

The resulting head formula in TNF can be viewed as an expression tree.

**Note**: Step 4 has not been implemented because of the difficulty associated with keeping track of the variables for every subformula and since this step has no strong justification and no test cases. The original rationale was that Step 4 can reduce the number of Skolem functions in rule bodies. However, Step 4 usually just replaces Skolem functions with universal quantifiers, which is equally useless. We need to study this issue further.

Note: While of course *naf* (and weak implication) cannot appear in a head, we can also speak of TNF for body formulas in which *naf* does appear, as long as it appears only in literals (i.e., before an atom or *neg* of an atom).

C. OH-C: Generation of Cranial Rules

Let Q be the input rule, having the form:

@{tq} HQ :- BQ.

The input rule's head HQ is already in TNF. As we will see, for each rule outputted by OH-C, that rule's head is a simper formula in TNF as well; that keeps things well defined when OH-C is called recursively or iteratively on the output of the earlier applications of OH-C.

1. The first and main work of OH-C is to transform the head, HQ.

Consider HQ's expression tree, and in particular its root node NR.

There are 5 cases, each corresponding to a distinct type of node.

1. If NR is a conjunction (A), then *split* it. In splitting, a head-formula conjunction

(H1 and ... and Hn)

is transformed into a set of n rules:

{Hi. | i=1,...,n}.

If NR is a universal quantifier (F), then *drop* it after renaming the variables quantified by that quantifier. Note that any such variable would appear in HQ only, not in BQ.

For instance,

forall(?x,?y)^(G(?x,?w),exists(?x)^H(?x,?y))

is transformed into

G(?newvar1,?w), exists(?x)^H(?x,?newvar2).

The set of implicitly quantified variables must be kept track of for use in skolemization (case (d.) below). The variable occurrences that are not explicitly quantified, like ?w in the above example, are also considered to be implicitly universally quantified. Thus, in this example, the list of implicitly universally quantified variables that results after dropping the above *forall* is <?newvar1,?newvar2,?w>.

1. If NR is a disjunction (O), then *directionalize* it. In directionalization, a head-formula disjunction

(G1 or ... or Gn)

is transformed into a set of n directional-variant rules:

{Gi :- bigand\_{j ≠ i | j=1,...,n} neg Gj. | i=1,...,n}.

We call the bodies of those rules: "cranial".

E.g., (G1 or G2 or G3) is transformed into the set of rules:

G1 :- neg G2 and neg G3.

G2 :- neg G1 and neg G3.

G3 :- neg G1 and neg G2.

If the disjunction contains complementary literals, issue a warning to the user. Duplicate literals are removed. This step is not implemented because it requires extra scans, but is not likely to be very common in practice.

After directionalization (at any later time): in each directional-variant, apply OB to the "cranial" body. When consolidating the "cranial" body with the body of the rule input to this step of OH, put the "cranial" after the "input" body. The rationale for this placement of the cranial body is that rules are typically written with the idea that the input-body literals will bind the variables that appear in the head, i.e., in cranial bodies.

Note that Ergo does not permit sensors or built-ins in the rule heads—this kind of literals can appear only in the bodies of the input rules. Therefore, sensors and built-ins will not occur in the heads of transformed rules.

Output "auxiliary" information that the "cranial" body subformulas are **refutable** body formulas (rbf).

1. If NR is an existential quantifier (∃), then skolemize it. In skolemization, the quantifier is omitted and instead the quantified variable is replaced by a skolem term. The argument tuple of the skolem term is the tuple of all currently free variables, which—recall—are implicitly universally quantified (their forall’s have been dropped earlier, in Step 1.b). For instance, in the example of case (b) above, this tuple of implicit universal variables is <?newvar1,?newvar2,?w>. Therefore exists(?x)^H(?x,?newvar2) in that example is transformed by replacing the existential variable ?x with a function term as follows:

H(sk(?newvar1,?newvar2,?w)).

where sk is a newly introduced skolem function symbol.

Keep in mind that skolemization is an ***unsound*** operation: the skolemized formula implies the original, but not vice versa. For example: foo(sk1) |= ∃X.foo(X), but ∃X.foo(X) |≠ foo(sk1). Therefore, it is desirable to keep the number of skolemizations to a minimum.

**Benjamin comment**: When one adds outermost existential quantification on the skolem functions, corresponding to dropping (i.e., projecting away) all conclusions that mention skolem functions, then it may well be that skolemization is indeed sound.

We need to study this further. Restrictions may be needed to show soundness and also completeness/equivalence.

An example is the following.

exist(?x)^(big(?x) and dog(?x)).

canine(?y) :- dog(?y).

Let sk be the skolem from the first omni. The entailment canine(sk) is unsound and dropped, while the entailment exist(?z)^(big(?z) and canine(?z)) is sound and not dropped.

**Michael’s comment**: This is certainly true in some cases, but it is not sound in general. This is a useless statement unless you come up with the actual conditions. For almost any kind of non-sequitur one can come up with restrictions where that non-sequitur happens to be true. For a counterexample, just look at the “unconvincing” example that you sent to me. There is no equivalence. In fact, the first method, which you thought is desirable, is much worse than the second. It produces quite meaningless rules like exists(X)p(X) :- forall(Y) neg p(Y). Under some assumptions, this makes the converted formula inconsistent while the original and Method#2 are consistent.

1. If NR is a fully pushed *neg*, or NR is an atom, then leave the expression tree unchanged. I.e., if the formula corresponding to the expression tree is a literal, then leave that literal unchanged.

In each of the cases (a)-(e), an input head-formula HQ is thus transformed into a non-empty set S of immediate-output rules Si. In case (e), S is simply the set containing a single fact: {G }. In cases (a)-(e), each Si's head-formula is a proper sub-formula of G.

2. Next, OH-C's set of overall output rules is formed from the result of (1.), as follows.

Let the form of a rule in the output of (1.) be:

HSi :- BSi.

For each such rule, take the OH-C’s input rule’s tag and body to form the following rule:

@{tq} HSi :- BQ and BSi.

Recall that we heuristically place BSi at the tail of the new body because it is anticipated that all the new bindings will be the result of evaluating BQ.

D. Iterate OH-C recursively to Generate Unit-Head Cranial Rules

For each output rule of OH-C, iterate recursively to apply OH-C, until exhaustion. This generation is performed by recursing on rule heads, e.g., HSi above. The recursion finishes when the rule heads are unit literals. At that point, the output rules are all in NF1.

Note that OH-TNF is applied only once, before OH-C begins.

In terms of OH-C's cases (1.a)-(1.e): (1.e) is the base case of the recursion, while (1.a)-(1.e) are non-trivial recursive steps.

**6. Miscellaneous details and remarks**

A. Refutable Body Formulas and the Argumentation Theory

The refutable body formulas (rbf's) are the "cranial" formulas that are moved into the body from the head (via directionalization in OH). I.e., a body formula is refutable iff it arises from head-alone. This form of the auxiliary info generated in output of OT differs from the expectation of ATCO2 which is: facts about refutable body literals -- refutableBodyLiteral ("rbo") -- as in OT-NF2. See the discussion at the end of section 2.

TODO (Benjamin): ATCO needs to be revised accordingly to support NF1 and rbf's, rather than NF2 and rbo's. (MK: unlikely. ATCO has no business or other kind of justification.)

* Background: The meta info required for specifying rbo's is described in the preamble to the file *flora2/AT/prolog/atco\_init.P* in the FLORA-2 repository on flora.sourceforge.net.
  + Also see: There are many examples in files *flora2-testsuite/defeasible/atco2/\*.flr*
  + Also see: Benjamin's file notes-avoid-ibo-v2.txt for details about how irrefutable can be treated implicitly essentially as naf of refutable. (We don't want to generate and use explicit irrefutablebodylit facts; instead, there's a way to keep track of all body literals, and any body literal which is not (i.e., naf) refutablebodylit is (implicitly) irrefutablebodylit. I.e., Ergo and ATCO2 eliminate the need to generate explicit irrefutablebodylit facts.)

1. Sensors in rule heads

Conceptually there is a restriction that sensors (including built-ins) are not allowed to appear in rule heads, and Ergo issues syntactic errors in this case. Likewise, effectors are not allowed to appear in rule bodies. A key way either may arise is as the result of directionalization.

Benjamin believes it may be convenient for authoring, however, to permit sensors to appear in omniform rule heads, and to permit effectors to be mixed with disjunction in omniform rule heads, as long as it is understood that disallowed directional variants are to be dropped. For instance, it may be beneficial to allow the user write

large(number)(?X) <== (*integer*(?X) and ?x>100).

(*integer*/1 and >/2 each being a built-in sensor) and then let the system take a “corrective action” and change this to

large(number)(?X) :- integer(?X) and ?x>100.

There is thus an issue of how to reconcile good language design with authoring convenience.

The Ergo approach is as follows. We disallow violations and report them as errors .But in an authoring tool layer/level above that, e.g., in an adapter for Linguist, one may permit the above if found convenient, and then produce Ergo rules that obey the above restrictions with no head sensors. Such a "corrective" step would automatically filter out (i.e., drop) disallowed directional variants by dropping them, issue a warning for each such dropping, and perhaps solicit user editing in other ways as well.

A primary reason we disallow violations in manual authoring is that if one were to drop violating directional variants that would sweep the problem under the rug and leave the KE too often unawares. In practice, even warnings are typically ignored, especially by inexperienced users and especially when their number exceeds two or three.

Benjamin believes that for automated authoring tools (such as Linguist) the syntax could be relaxed. Michael is skeptical that this is a good idea even for automated conversion because (i) it is equally easy, if not easier, for the converters to produce code where sensors/built-ins are always in the rule bodies and (ii) even automated converters should not be allowed to produce meaningless output such as foo(?X) <==>*integer*(?X), as this indicates a modeling defect or a misunderstanding on the part of the converter’s author. Such a problem should not be swept under the rug since it will not be discovered without great difficulty.

C. DerivedFrom info -- additional bookkeeping info generation

Along the way, within various transformation steps at coarser and finer grain:

Add meta/annotation info in each post-transform rule to keep track of the derivedFrom relationship to the pre-transform rule.

This derivedFrom info is optional with respect to the semantics. But it will be very useful for debugging the code, and much of it will be quite useful for supporting explanation and knowledge debugging. *Practically speaking, in the implementation we will definitely want to do it.*

D. Reification

To support the omniformity properly in the overall language syntax: reification must be accordingly extended to permit omniform formulas. *Practically, this is important for Linguist/Textual-Logic to treat NL modals such as "can" and "should", that (not infrequently) appear in verb phrases.*

TODO later: specify this in detail.

E. Examples

See separate documents that give examples of the overall omni transform. E.g., the subdirectory named ***examples*** sent by Benjamin in various versions starting in late May 2013.

F. Effectors

TODO for fairly near future: add effectors to Ergo, and discuss/treat the potential for effectors to appear in the body, in subsection (B.) above.

G. Computational Complexity of OT

Our preliminary analysis indicates that OT's output size (and time required) is essentially worst-case polynomial in the size of the input, more specifically the following. It is worst-case quadratic, if the number of strong or weak equivalences per rule is bounded. The supra-linearity is driven by the numbers of existentials per rule head, along with the numbers of universals to their left and literals to their right. If the numbers of these are bounded per rule, then it is probably worst-case linear. Practically speaking, these numbers are indeed bounded *de facto*: to roughly 10 or so quantifiers, and 10-20 literals, in the case of omniform rules that arise from per-sentence text-based authoring such as in Linguist, since sentence length is bounded to roughly 30 words per sentence. In more detail: Head existentials nested within universals results in potentially quadratic expansion from skolem terms having to mention explicitly all the outer universal variables.

Interestingly, head disjunction does not drive supra-linearity, intrinsically, as we earlier supposed. The set of rules resulting from the directionalization step can be represented in size linear in the input rule. The "trick" is to define a set of macros that represent the different bodies in the result of directionalization, in the following manner.

Let the disjunctive head formula be (G1 or ... or Gm) , where each Gi is itself a formula.

Inductively define new atoms S1,...,Sm and T1,..., Tm as follows (i.e., atoms in newly introduced predicates):

S1 :- neg G1.

Tm:- neg Gm.

Si+1 :- Si and neg Gi+1. /\* for i=1..m-1 \*/

Sj-1 :- Tj and neg Gj-1. /\* for j=2..m \*/

Then the set of directional variants can be represented more concisely as:

{ Gi :- Si-1 and Ti+1. | i = 1..m}.

rather than less concisely as

{ Gi :- neg G1 and ... and neg Gi-1 and neg Gi+1 and ... and neg Gm | | i = 1..m}.

However, practically, we do not want to implement this "trick" initially, because we are avoiding macro introduction and because the number of head disjunctions in the input to OT is not likely to justify the development and runtime effort involved.

We have an idea for a second "trick" to keep the size blowup of the omni transform at worst-case linear, for the skolem terms aspect and thus for the overall transform. The idea is to "package up" tuples of variables (from the universals), via tuple/list variables or summary frame-ish predicate+variable combinations.

However, practically, we do not want to implement such a second "trick" (even if it works) initially, because it is

- not worth the development and justification hair for now and probably ever; and

- not worth it for size blowup until the quantifier count, with enough nesting and inner literals, is up at 10 or so, anyway.

- not practically a win from a size blowup viewpoint anyway, given the bounded length/complexity of omnis in practice that arise from English sentences in KA (as discussed earlier).

TODO later: check and refine the above analysis.