New Smasher – Algorithm, Examples and Results

Definitions

1. *Modifier*: an (et)et function
2. *Modifiee*: an et function
3. *Main Argument*: the argument of the *Modifiee* function.
4. Significant Applications: all application to the Main Argument.
5. Base Predicates: All constant names of predicates that appear in Significant Applications.
6. Free Applications: all applications to an argument that is not the Main Argument, and that are not embedded in a Significant Application.
7. New Predicate: A new predicate that is created as part of the smashing.
8. Content of Abstraction: The part after the variable introduction (P(x) in \x0. P(x)).
9. Constant sets:
    a. Quantifiers (Q): EXISTS, FORALL
    b. Scope-Sensitive-Elements (SSE): IMPLIES

Illustration:

`Consider Sentence 1: John deeply loves a girl`

1. Modifier: `deeply:(et)et`
2. Modifiee: `\x0:e.(EXISTS:(et)t (\x1:e.((AND:ttt (girl:et x1:e)) ((loves:eet x1:e) x0:e))))`
3. Main Argument: x0
4. Significant Applications: {`((loves:eet x1:e) x0:e)`}
5. Base Predicates: {loves}
6. Free Applications: {`(girl:et x1:e)`}
7. New Predicate: deeply_loves

Algorithm Description

Preparation Steps

1. Find the variable with the highest id among those in the Modifier and the Modifie.
   Illustration: In Sentence 1, it is 'x1'.
2. Add an explicit Main Argument if missing
   In cases where the Modifiee is a simple constant, add an abstraction and application of a new fresh variable to it. This step is performed only for having uniformity in the rest of the code.
   Illustration: if the Modifiee is *man:et*, and the Modifier includes the variable x4 as the variable with the highest id, the Modifiee becomes: *\x5:e. (man:et x5:e)*
3. Recognize the Main Argument.
   Illustration: 'x5' in the Modifiee *\x5:e. (man:et x5:e)*.

Transformation Steps

1. Front all quantifiers while keeping their relative order and alpha-convert all bound variables from original indexes to fresh ones. Fronting of quantifiers over conjunctions, disjunctions and implications is safe in the sense of generating a logically-equivalent expression (Reference).[1] That is, if x is not free variable in ϕ:
   1. $((\forall\ x\ (\phi \rightarrow \psi)) \leftrightarrow (\phi \rightarrow (\forall\ x\ \psi)))$
      $((\exists\ x\ (\phi \rightarrow \psi)) \leftrightarrow (\phi \rightarrow (\exists\ x\ \psi)))$
   2. $((\forall\ x\ (\phi \vee \psi)) \leftrightarrow (\phi \vee (\forall\ x\ \psi)))$
      $((\exists\ x\ (\phi \vee \psi)) \leftrightarrow (\phi \vee (\exists\ x\ \psi)))$
   3. $((\forall\ x\ (\phi \wedge \psi)) \leftrightarrow (\phi \wedge (\forall\ x\ \psi)))$
      $(\exists\ x\ (\phi \wedge \psi)) \leftrightarrow (\phi \wedge (\exists\ x\ \psi))$

   This step results in having a Modified Modifiee in which all quantifiers are fronted.
2. Collect all Significant Applications and create the New Predicate.
   The purpose of this step is to have a list of all Significant Applications for creating the name of the new predicate and for parameterizing it. The name is determined based on the names of the Base Predicates and the Modifier name. The parameterization done based on aggregation of parameters of the Base Predicates and of the Modifier.
3. Add the New Predicate to the Modified Modifiee by means of conjunction under the quantifiers.
4. Return the Modified Modifiee as the smashing result.

[1] Note that in the beginning I thought the fronting over implication is not safe but I was wrong.

Examples

1. Sentence: John deeply [loves a girl]
   a. Modifier: `deeply:(et)et`
   b. Modifiee:
      `\x0:e.(EXISTS:(et)t (\x1:e.((AND:ttt (girl:et x1:e)) ((loves:eet x1:e) x0:e))))`
   c. Base Predicates: {loves}
   d. Smashing Result:
      `\x0:e.(EXISTS:(et)t (\x2:e.((AND:ttt ((AND:ttt (girl:et x2:e)) ((loves:eet x2:e) x0:e))) ((deeply_loves:eet x2:e) x0:e)))`

   Sentence in FOL:
   `(exists x0 (girl(x0) & loves(x0, John)) & exists x1 ((girl(x1) & loves(x1, John)) & deeply_loves(x1, John))).`

2. Sentence: John [[reads a book] in the library]
   a. Modifier: `in:e(et)et c1:e`
   b. Modifiee:
      `\x0:e.(EXISTS:(et)t (\x1:e.((AND:ttt (book:et x1:e)) ((reads:eet x1:e) x0:e))))`
   c. Base Predicates: {`reads`}
   d. Smashing Result:
      `\x0:e.(EXISTS:(et)t (\x2:e.((AND:ttt ((AND:ttt (book:et x2:e)) ((reads:eet x2:e) x0:e))) (((in_reads:eeet c1:e) x2:e) x0:e))))`

   Sentence in FOL:
   `(exists x0 (book(x0) & reads(x0, John)) & exists x1 ((book(x1) & reads(x1, John)) & in_reads(c1, x1, John))).`

3. Sentence: John is tall [Dutch man]
   a. Modifier: `short:(et)et`
   b. Modifiee: `\x0:e.((AND:ttt (man:et x0:e)) (dutch:et x0:e))`
   c. Base predicates: {man, dutch}
   d. Smashing result:
      `\x0:e.((AND:ttt ((AND:ttt (man:et x0:e)) (dutch:et x0:e))) (short_man_dutch:et x0:e))`

   Sentence in FOL:
   `exists x0 (((man(x0) & dutch(x0)) & ((man(x0) & dutch(x0)) & short_man_dutch(x0))) & x0=jan).`

4. Sentence: John [[loves every boy who admires every girl] during XMAS]
    a. Modifier: `during:e(et)et XMAS:e`
    b. Modifiee:
    ```
    \x0:e.(FORALL:(et)t (\x1:e.((IMPLIES:ttt ((AND:ttt (boy:et x1:e))
    (FORALL:(et)t (\x2:e.((IMPLIES:ttt (girl:et x2:e)) ((admires:eet x2:e)
    x1:e)))))) ((loves:eet x1:e) x0:e))))
    ```
    c. Base Predicates: {loves}
    d. Smashing Result:
    ```
    \x0:e.(FORALL:(et)t (\x3:e.(FORALL:(et)t (\x4:e.((AND:ttt ((IMPLIES:ttt
    ((AND:ttt (boy:et x3:e)) ((IMPLIES:ttt (girl:et x4:e)) ((admires:eet
    x4:e) x3:e)))) ((loves:eet x3:e) x0:e))) (((during_loves:eeet XMAS:e)
    x3:e) x0:e))))))
    ```
    Sentence in FOL:
    ```
    (all x0 ((boy(x0) & all x1 (girl(x1) -> admires(x1, x0))) -> loves(x0,
    John)) & all x2 all x3 (((boy(x2) & (girl(x3) -> admires(x3, x2))) ->
    loves(x2, John)) & during_loves(XMAS, x2, John))).
    ```

Supported Inferences

Text	Hypothesis
Jan sat and ate	Jan sat / Jan ate
Jan is a short man	Jan is a man / *Jan is short
Jan is a Dutch man	Jan is Dutch / John is a man
Jan is a short Dutch man	Jan is a man / John is Dutch / John is Dutch man / *Jan is a short man / *Jan is a short Dutch / *Jan is short
John is a fat tall man	Jan is a man / John is a tall man / *Jan is a fat man / *Jan is a fat tall / *Jan is fat / *Jan is tall
John extremely loves the girl	Jan loves the girl/ Jan loves the girl / John extremely loves a girl
A Korean ate the eggs and quickly ate the meatballs	A Korean ate the eggs and ate the meatballs / * A Korean [quickly ate the eggs] and ate the meatballs
John saw the [girl from the lab]	* John saw the [girl from Iraq]
A man extremely [adores Jane and loves the girl]	A man adores Jane / A man loves the girl / A man adores Jane and loves the girl / A man extremely [adores the girl and loves the girl] / A man extremely [adores Jane and loves Jane]
John kisses a girl on every day	John kisses a girl on every nice day
John kisses Mary on every day	Some man kisses a girl on every nice day
John loves XMAS on every day in every year	John loves XMAS on Monday in every year

Text in Red: Invalid inferences that the original smasher supports and the new one doesn't (correctly). The original supported them because it doesn't parameterize.

Text in Green: Intuitive but logically unlicensed inferences that the new Smasher supports. These are inferences of the kind Mod(Set1)→Mod(Set2) whereby Set1⊆Set2, as in: extremely [kissed Mary] → extremely [kissed a girl].