Harnessing Projection

A Formal Implementation of Projective Discourse Representation Theory

Noortje Venhuizen
n.j.venhuizen@rug.nl

Harm Brouwer
harm.brouwer@rug.nl

university of groningen

CLIN 24

In this talk

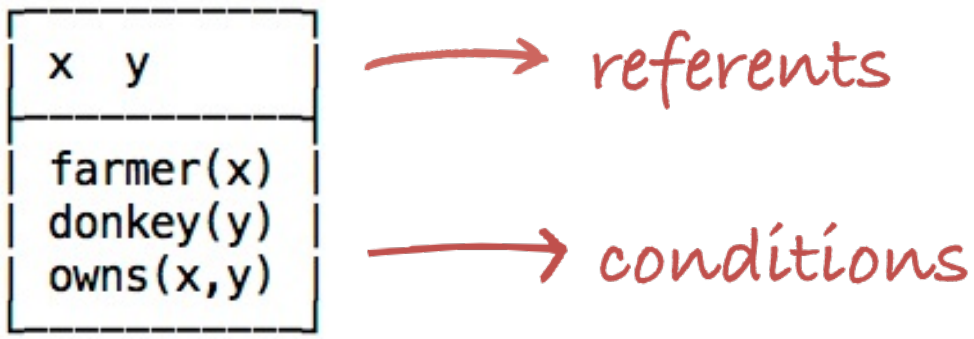
- What is Projective Discourse Representation Theory?
- We present PDRT-SANDBOX: an NLP Haskell library that implements the PDRT formalism (as well as DRT).

Discourse Representation Theory

A representational, dynamic formalism for representing meaning.

Kamp (1981)

A farmer owns a donkey.

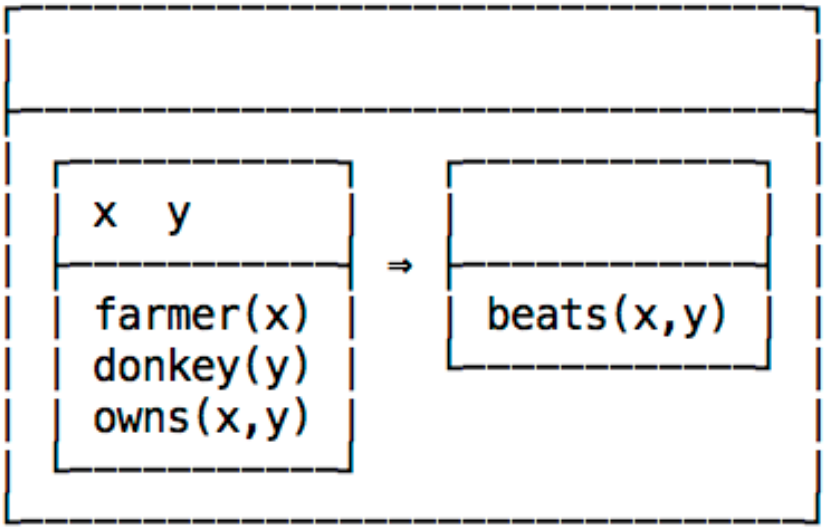


Discourse Representation Theory

A representational, dynamic formalism for representing meaning.

Kamp (1981)

If a farmer owns a donkey, he beats it.



DRT's pros

Wide coverage: anaphora, tense, presupposition

(van der Sandt, 1992; Kamp & Reyle, 1993)

Model-theoretic interpretation: FOL translation

(Kamp & Reyle, 1993; Bos, 2004)

"Compositional": Montague Semantics

(cf. Muskens, 1996)

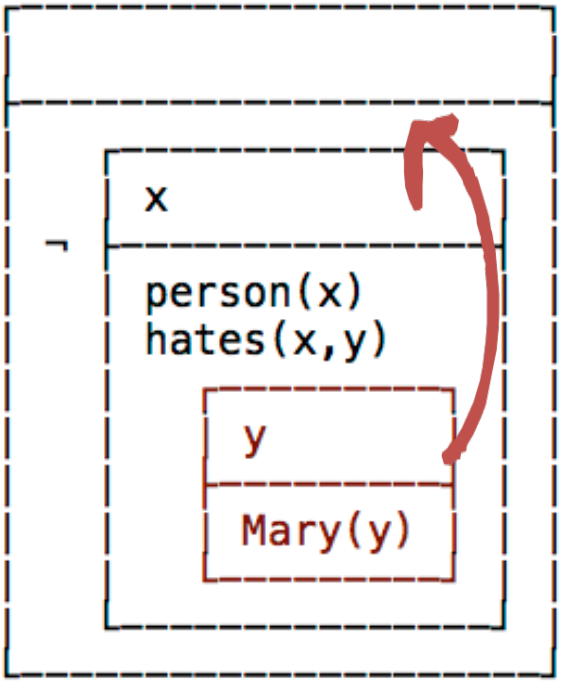
Problem: these properties do not always combine very well

Projection vs. Compositionality

A presupposition is a piece of backgrounded information.

Projection refers to the property of being indifferent to semantic operators.

Nobody hates Mary. → Mary exists



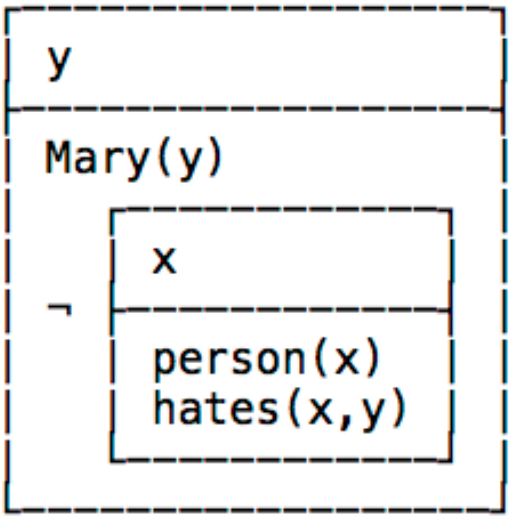
van der Sandt (1992): In DRT, presuppositions are moved in a second step of processing

Projection vs. Compositionality

A presupposition is a piece of backgrounded information.

Projection refers to the property of being indifferent to semantic operators.

Nobody hates Mary. → Mary exists



Sandt (1992): In DRT,

How to reconcile this with a compositional, Montague-style construction procedure?

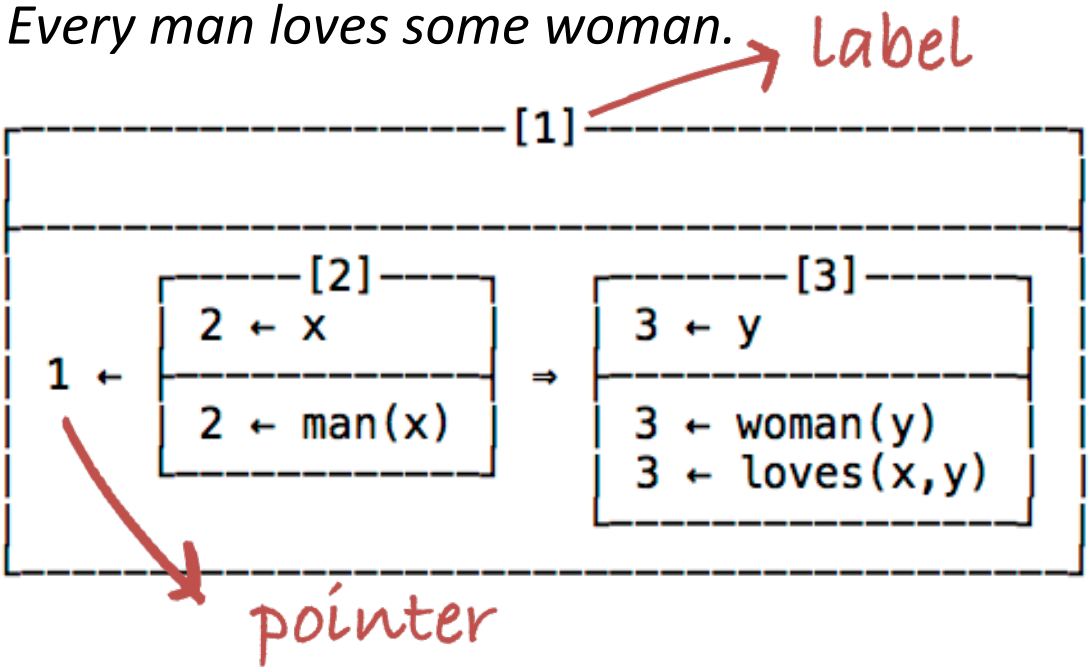
A theoretical solution

Projective Discourse Representation Theory:

Adding Projection Pointers to DRSs

(Venhuizen, Bos, & Brouwer, 2013)

Every man loves some woman.



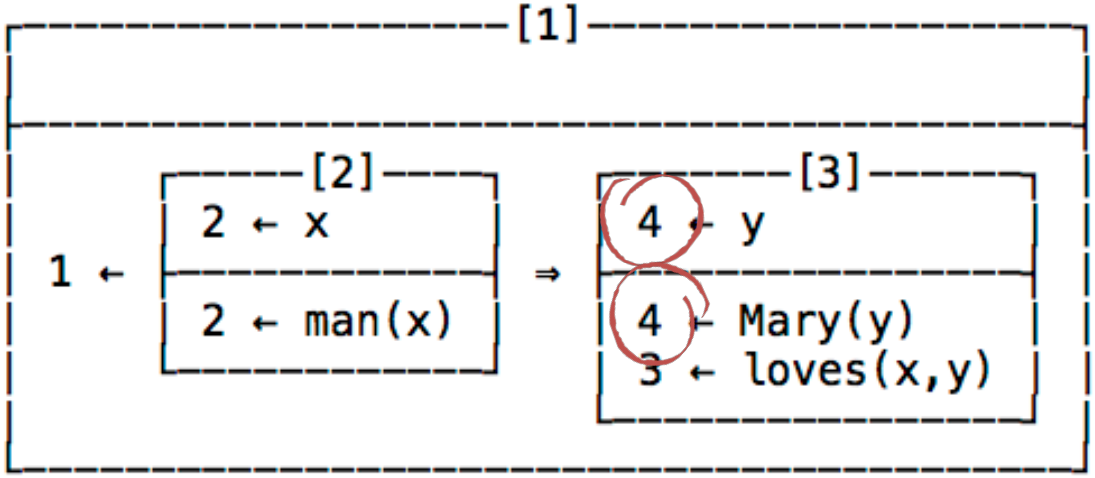
A theoretical solution

Projective Discourse Representation Theory:

Adding Projection Pointers to DRSs

(Venhuizen, Bos, & Brouwer, 2013)

Every man loves Mary.



But then, what about:

- Merging discourse structures?
- Anaphoric binding?
- Accessible contexts?
- Unresolved (P)DRSs?
- Semantic interpretation?

Let's implement it!

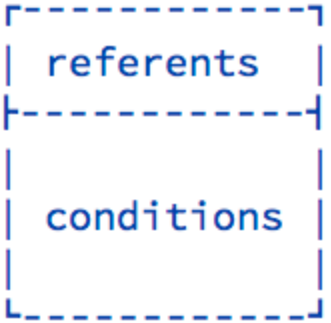
Background: DRS definitions

Basic DRS:

<{DRS referents},{DRS conditions}>

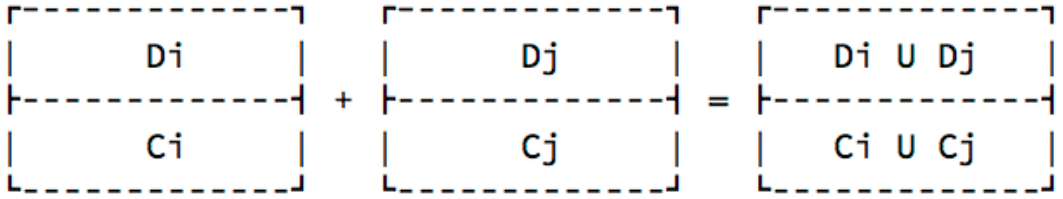
x, y

john(x), loves(x,y),
¬ DRS, $DRS_1 \Rightarrow DRS_2$

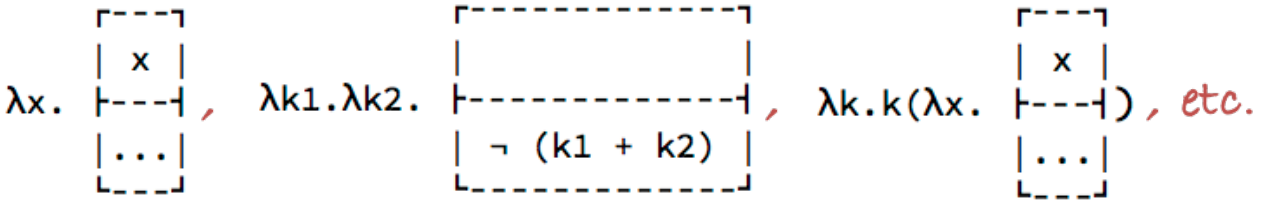


Complex DRS:

- Merge

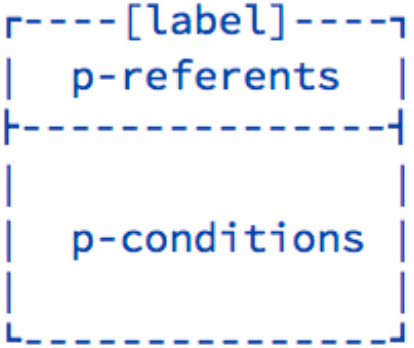


- Lambda DRS



Basic PDRSs

Basic PDRS:



`<label,{p-referents},{p-conditions}>`

where:

p-referent:	`<pointer, DRS referent>`
p-condition:	`<pointer, PDRS condition>`
pointer, label:	`projection variables` 1, 2

asserted content:	label = pointer
projected content:	label ≤ pointer → accessible context
presupposed content:	pointer occurs free in the global PDRS

Accessibility of contexts

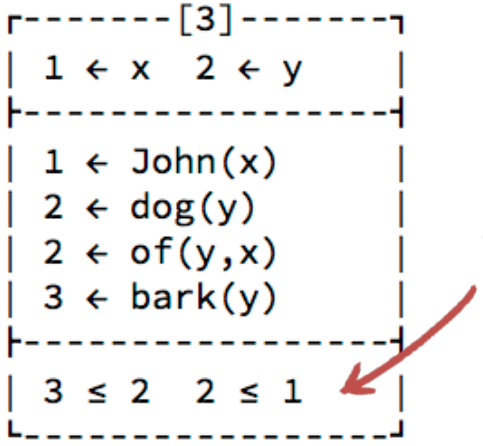
Context C1 subordinates context C2 (C2 ≤ C1) iff

(i) C2 is embedded in C1

(ii) C2 is the antecedent of C2

(iii) C2 is a MAP with respect to C1

John's dog barks



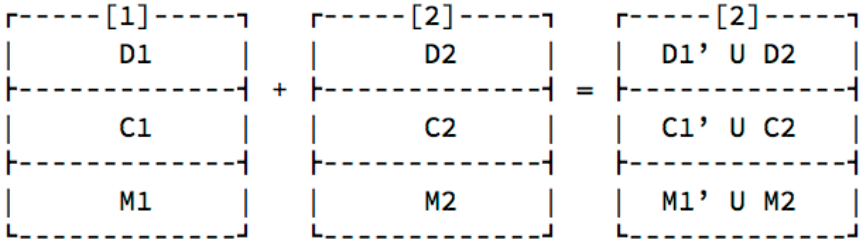
Minimally Accessible
Projection context (MAPs)

Revised definition for basic PDRSs:
<label,{MAPs},{p-referents},{p-conditions}>

Complex PDRSs: Merge

Assertive Merge

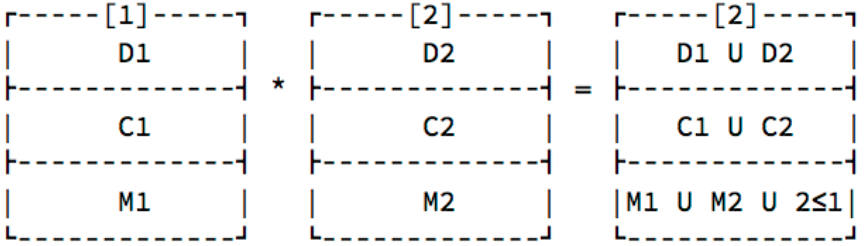
keep all asserted content from $PDRS_1$ asserted



where: $PDRS_1' = \alpha\text{-convert}\ (l_1, l_2)\ PDRS_1$

Projective Merge

project all asserted content from $PDRS_1$



Caution: prevent accidental variable binding!

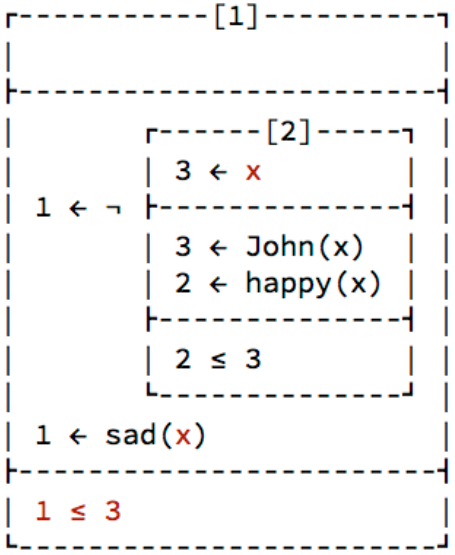
α-convert all bound variables in context PDRS

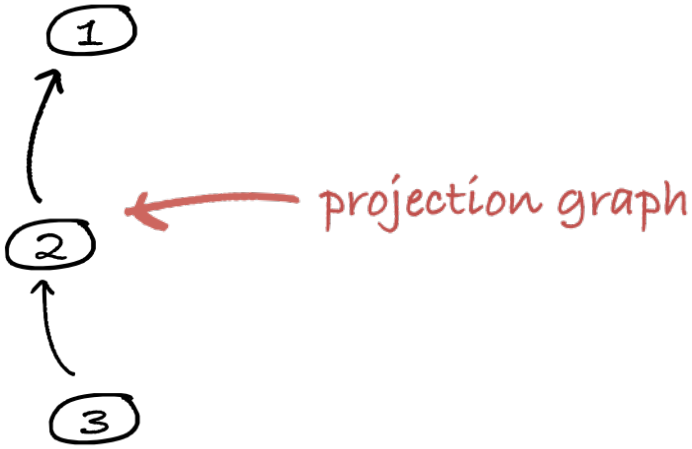
define free and bound variables in PDRT

Free and bound variables in PDRT

Binding of pointers in PDRT defined just like referents in DRT: determined by accessible PDRSs.

It is not the case that John is happy. He is sad.

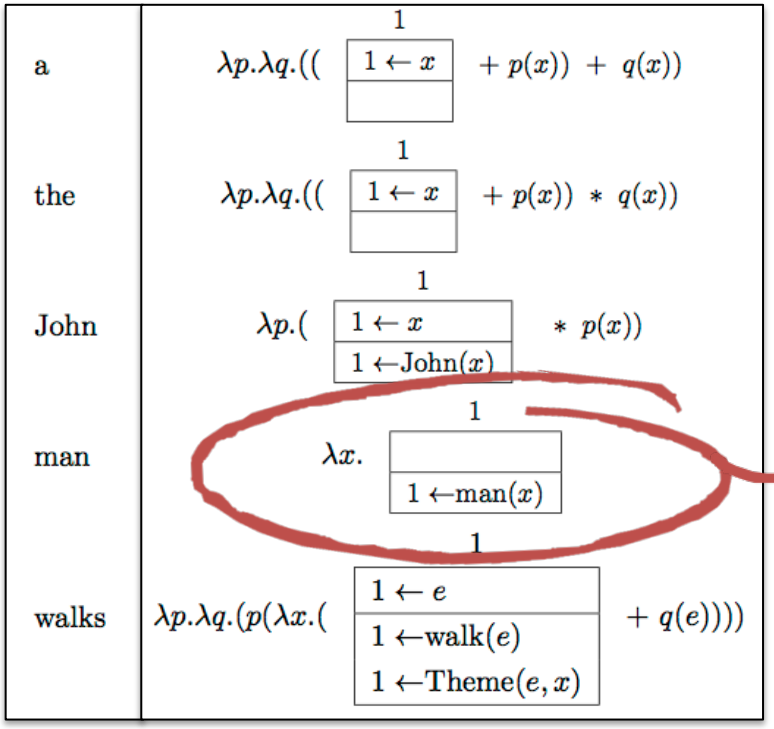




What about referents in PDRT? Non-hierarchical binding through projection!

Complex PDRSs: Unresolved structures

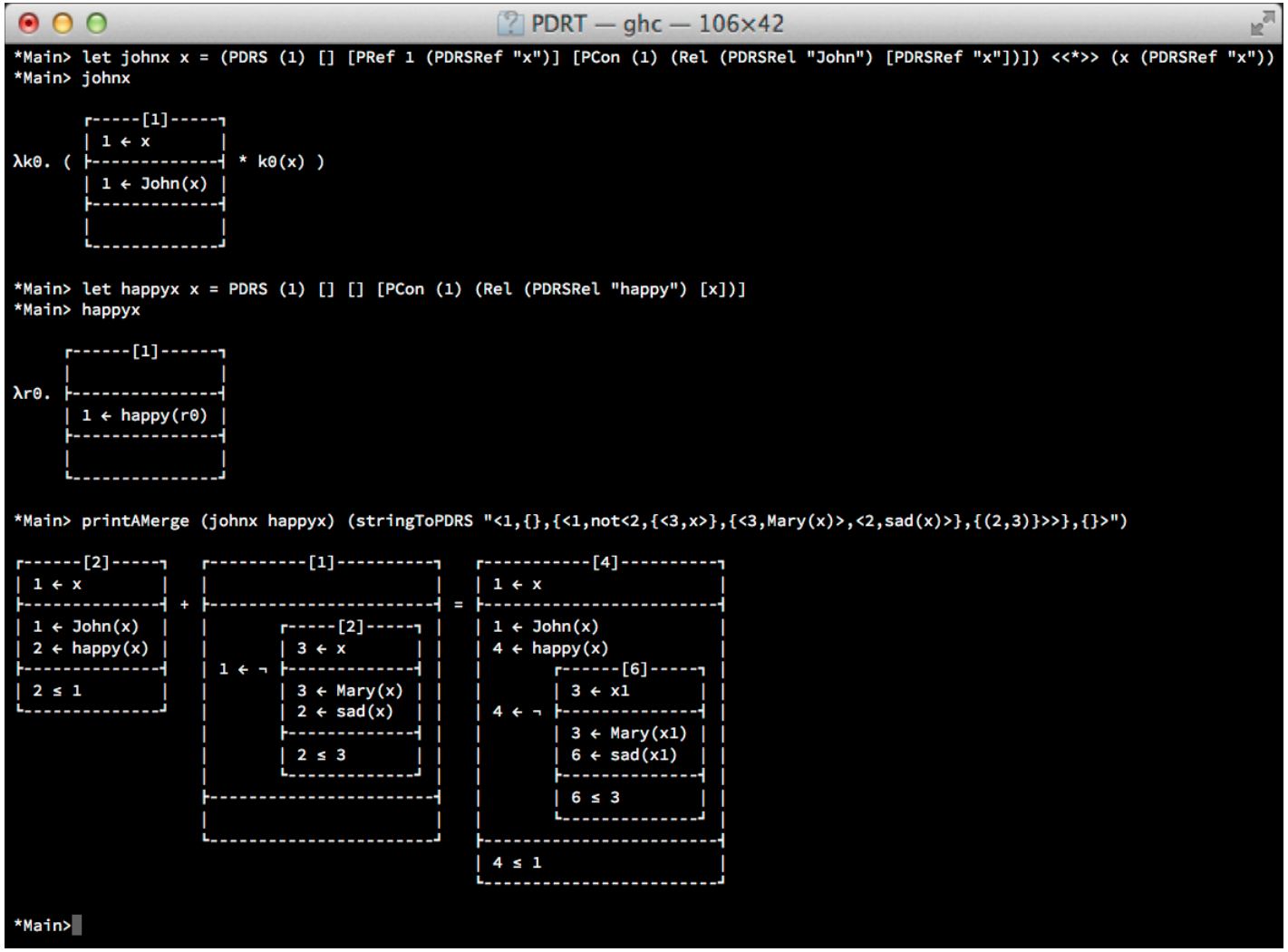
Following Muskens (1996) we can define unresolved PDRSs as lambda terms.




```
man :: PDRSRef -> PDRS
man x = PDRS 1
          []
          []
          [PCon 1 (Rel (PDRSRel man) [x])]
```

Functions in Haskell are already defined as lambda terms!

PDRT-SANDBOX in action!



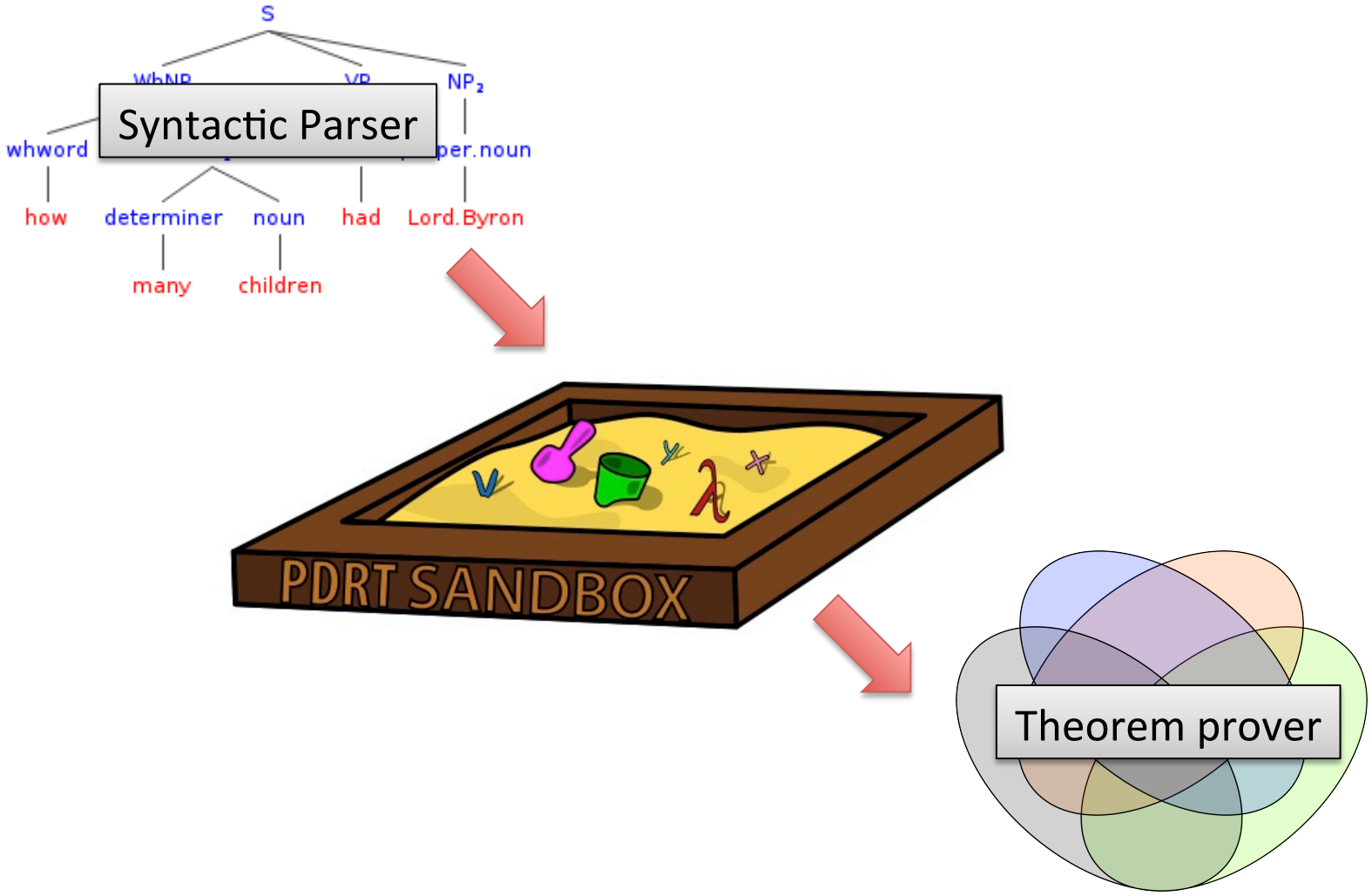
PDRT-SANDBOX

A full-fledged NLP Haskell library that allows for:

- defining your own PDRSs and DRSs
- showing and manipulating structures
- combining unresolved structures
- translating PDRSs into DRSs and FOL formulas
- various input/output formats
- and much more…

Ask for a demo!

Combining PDRT-SANDBOX with other tools



PDRT to DRT and FOL Translations

```
Prelude Data.PDRS> let notjohnhappy = PDRS (1) [] [] [PCon (1) (Neg (PDRS (2) [(2,3)] [PRef 3 (PDRSRef "x")]
[PCon (3) (Rel (PDRSRel "John") [PDRSRef "x"]),PCon (2) (Rel (PDRSRel "happy") [PDRSRef "x"])]))]
Prelude Data.PDRS> notjohnhappy

┌-----------[1]----------┐
|                        |
├------------------------┤
|       ┌------[2]-----┐ |
|       | 3 ← x        | |
| 1 ← ¬ ├--------------┤ |
|       | 3 ← John(x)  | |
|       | 2 ← happy(x) | |
|       ├--------------┤ |
|       | 2 ≤ 3        | |
|       └--------------┘ |
├------------------------┤
|                        |
└------------------------┘

Prelude Data.PDRS> pdrsToDRS notjohnhappy

┌-----------------┐
| x               |
├-----------------┤
|    ┌----------┐ |
|    |          | |
| ¬  ├----------┤ |
|    | happy(x) | |
|    └----------┘ |
| John(x)         |
└-----------------┘

Prelude Data.PDRS> pdrsToFOL notjohnhappy

∃x(¬happy(w,x) ∧ John(w,x))
```

Output formats

```
Prelude Data.PDRS> let notjohnhappy = PDRS (1) [] [] [PCon (1) (Neg (PDRS (2) [(2,3)] [PRef 3 (PDRSRef "x")] [PCon (3)
(Rel (PDRSRel "John") [PDRSRef "x"]),PCon (2) (Rel (PDRSRel "happy") [PDRSRef "x"])]))]
Prelude Data.PDRS> notjohnhappy

┌-----------[1]----------┐
|                        |
├------------------------┤
|       ┌------[2]-----┐ |
|       | 3 ← x        | |
| 1 ← ¬ ├--------------┤ |
|       | 3 ← John(x)  | |
|       | 2 ← happy(x) | |
|       ├--------------┤ |
|       | 2 ≤ 3        | |
|       └--------------┘ |
├------------------------┤
|                        |
└------------------------┘

Prelude Data.PDRS> Linear notjohnhappy

1:[|(1,¬2:[(3,x)|(3,John(x)),(2,happy(x))|(2,3)])|]

Prelude Data.PDRS> Set notjohnhappy

<1,{},{},{(1,¬<2,{(2,3)},{(3,x)},{(3,John(x)),(2,happy(x))}>)}>

Prelude Data.PDRS> Debug notjohnhappy

PDRS (1) [] [] [PCon (1) (Neg (PDRS (2) [(2,3)] [PRef 3 (PDRSRef "x")] [PCon (3) (Rel (PDRSRel "John") [PDRSRef
"x"]),PCon (2) (Rel (PDRSRel "happy") [PDRSRef "x"])]))]

Prelude Data.PDRS> show $ Debug notjohnhappy
"\nPDRS (1) [] [] [PCon (1) (Neg (PDRS (2) [(2,3)] [PRef 3 (PDRSRef \"x\")] [PCon (3) (Rel (PDRSRel \"John\") [PDRSRef
\"x\"]),PCon (2) (Rel (PDRSRel \"happy\") [PDRSRef \"x\"])]))]\n"
Prelude Data.PDRS> writeFile “example1.hs” (show $ Debug notjohnhappy)
```

Lexical semantics

```
Prelude Data.PDRS> let a p q = AMerge (AMerge (PDRS 1 [] [PRef 1 (PDRSRef "x")] []) (p (PDRSRef "x1"))) (q (PDRSRef "x"))
Prelude Data.PDRS> a

            ┌──[1]──┐
            | 1 ← x |
λk0.λk1. ( ( ├───────┤ + k0(x) ) + k1(x) )
            |       |
            ├───────┤
            |       |
            └───────┘

Prelude Data.PDRS> let man x = PDRS 1 [] [] [PCon 1 (Rel (PDRSRel "man") [x])]
Prelude Data.PDRS> man

     ┌─────[1]─────┐
     |             |
λr0. ├─────────────┤
     | 1 ← man(r0) |
     ├─────────────┤
     |             |
     └─────────────┘

Prelude Data.PDRS> let walks p q = (p (\x -> AMerge (PDRS 1 [] [PRef 1 (PDRSRef "e")] [PCon 1 (Rel (PDRSRel "walks") [PDRSRef
"e"]),PCon 1 (Rel (PDRSRel "Theme") [PDRSRef "e",x])])) (q (PDRSRef "e")))
Prelude Data.PDRS> (walks (a man))

         ┌──[1]──┐   ┌─────[1]─────┐       ┌───────[1]───────┐
         | 1 ← x |   |             |       | 1 ← e           |
λk0. ( ( ├───────┤ + ├─────────────┤ ) + ( ├─────────────────┤ + k0(e) ) )
         |       |   | 1 ← man(x)  |       | 1 ← walks(e)    |
         ├───────┤   ├─────────────┤       | 1 ← Theme(e,x)  |
         |       |   |             |       ├─────────────────┤
         └───────┘   └─────────────┘       |                 |
                                           └─────────────────┘
```

Projection Table/Graph

```
Prelude Data.PDRS> example

┌---------------[1]---------------┐
| 8 ← x                           |
├---------------------------------┤
| 8 ← P(x)                        |
|       ┌-----[2]----┐            |
|       | 9 ← y      |            |
| 1 ← ◇ ├------------┤            |
|       | 2 ← Q(y)   |            |
|       | 9 ← R(x,y) |            |
|       ├------------┤            |
|       | 2 ≤ 9      |            |
|       └------------┘            |
|     ┌----[4]---┐   ┌----[5]---┐ |
|     | 4 ← y    |   |          | |
| 1 ← ├----------┤ ⇒ ├----------┤ |
|     | 4 ← M(x) |   | 5 ← N(x) | |
|     ├----------┤   | 4 ← N(y) | |
|     |          |   ├----------┤ |
|     └----------┘   |          | |
|                    └----------┘ |
├---------------------------------┤
| 1 ≤ 8                           |
└---------------------------------┘

Prelude Data.PDRS> projectionTable example

[Type]  [Content]  [Intro st]  [Proj. st]
Ref     x          1           8
Con     P(x)       1           8
Con     ◇ 2        1           1
Ref     y          2           9
Con     Q(y)       2           2
Con     R(x,y)     2           9
Con     4 ⇒ 5      1           1
Ref     y          4           4
Con     M(x)       4           4
Con     N(x)       5           5
Con     N(y)       5           4

Prelude Data.PDRS> projectionGraph example
array (1,9) [(1,[8,1]),(2,[9,2,1]),(3,[]),(4,[4,1]),(5,[5,4]),(6,[]),(7,[]),(8,[]),(9,[])]
```

Boxer and PDRT-SANDBOX: A happy marriage?

```
p264344@zardoz:~$  curl -d'The sloth moved very slowly.' http://localhost:7777/boxer?semantics=pdrs
%%% This output was generated by the following command:
%%% /home/p262301/dev/candc/bin/boxer --stdin --semantics pdrs
%%% The sloth moved very slowly .
id(1,1).
sem(1,[1001:[tok:'The',pos:'DT',lemma:the,namex:'Of],1002:[tok:sloth,pos:'NN',lemma:sloth,namex:'O'],1003:[tok:moved,pos:'VBD',lemma:move,namex:'O'],1004:
[tok:very,pos:'RB',lemma:very,namex:'O'],1005:[tok:slowly,pos:'RB',lemma:slowly,namex:'O'],1006:[tok:'.',pos:'.',lemma:'.',namex:'O']],_G1867:drs([_G1483:
[1001]:_G1508,_G1867:[]:_G1892],[_G1483:[1002]:pred(_G1508,sloth,n,0),_G1867:[1003]:pred(_G1892,move,v,0),_G1867:[]:role(_G1892,_G1508,agent,1),_G1867:
[1005]:pred(_G1892,slowly,a,0),_G1867:[1004]:pred(_G1892,very,a,0)])).

…

Prelude Data.PDRS> let sloth = boxerToPDRS "sem(1,[1001:[tok:'The',pos:'DT',lemma:the,namex:'O'],1002:[tok:sloth,pos:'NN',lemma:sloth,namex:'O'],1003:
tok:moved,pos:'VBD',lemma:move,namex:'O'],1004:[tok:very,pos:'RB',lemma:very,namex:'O'],1005:[tok:slowly,pos:'RB',lemma:slowly,namex:'O'],1006:
[tok:'.',pos:'.',lemma:'.',namex:'O']],_G1867:drs([_G1483:[1001]:_G1508,_G1867:[]:_G1892],[_G1483:[1002]:pred(_G1508,sloth,n,0),_G1867:[1003]:
pred(_G1892,move,v,0),_G1867:[]:role(_G1892,_G1508,agent,1),_G1867:[1005]:pred(_G1892,slowly,a,0),_G1867:[1004]:pred(_G1892,very,a,0)]))."
Prelude Data.PDRS> sloth

┌--------[1]--------┐
| 2 ← x1  1 ← x2    |
├-------------------┤
| 2 ← sloth(x1)     |
| 1 ← move(x2)      |
| 1 ← Agent(x2,x1)  |
| 1 ← slowly(x2)    |
| 1 ← very(x2)      |
├-------------------┤
|                   |
└-------------------┘

Prelude Data.PDRS> :t sloth
sloth :: PDRS
```

Why Haskell?


CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?