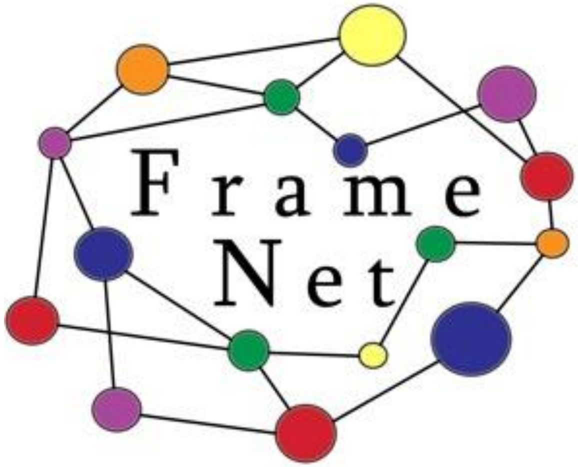
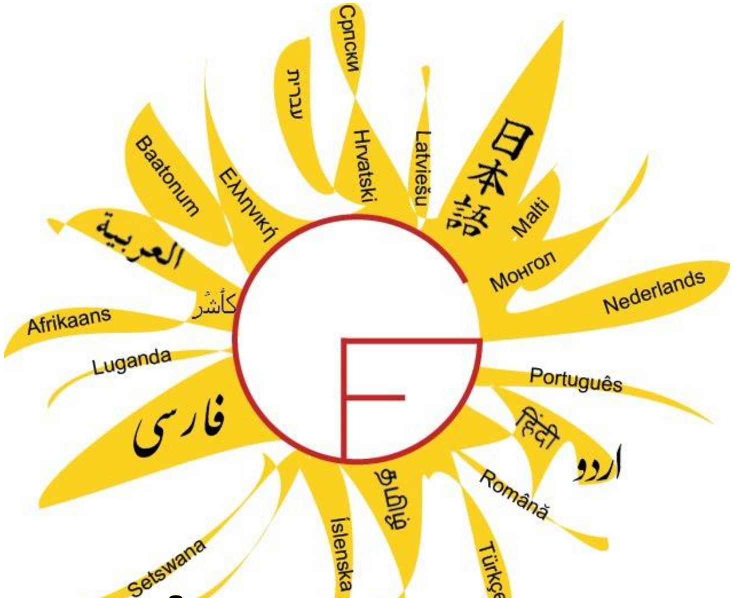




A Frame Semantic Abstraction Layer to the GF Resource Grammar Library

Normunds Grūzītis
Institute of Mathematics and Computer Science
University of Latvia

Third GF Summer School: Frontiers of Multilingual Technology: Scaling Up Grammatical Resources
Frauenchiemsee, Bavaria, Germany
18th–30th August, 2013

Outline

- A brief introduction to FrameNet
- FrameNet as a semantic API to GF RGL
  - For GF application grammar developers
    - Case-study: MOLTO Phrasebook
- A generalized FrameNet application grammar
  - For semantic parsing (semantic role labeling)
  - For natural language generation (from FrameNet-annotated knowledge bases)

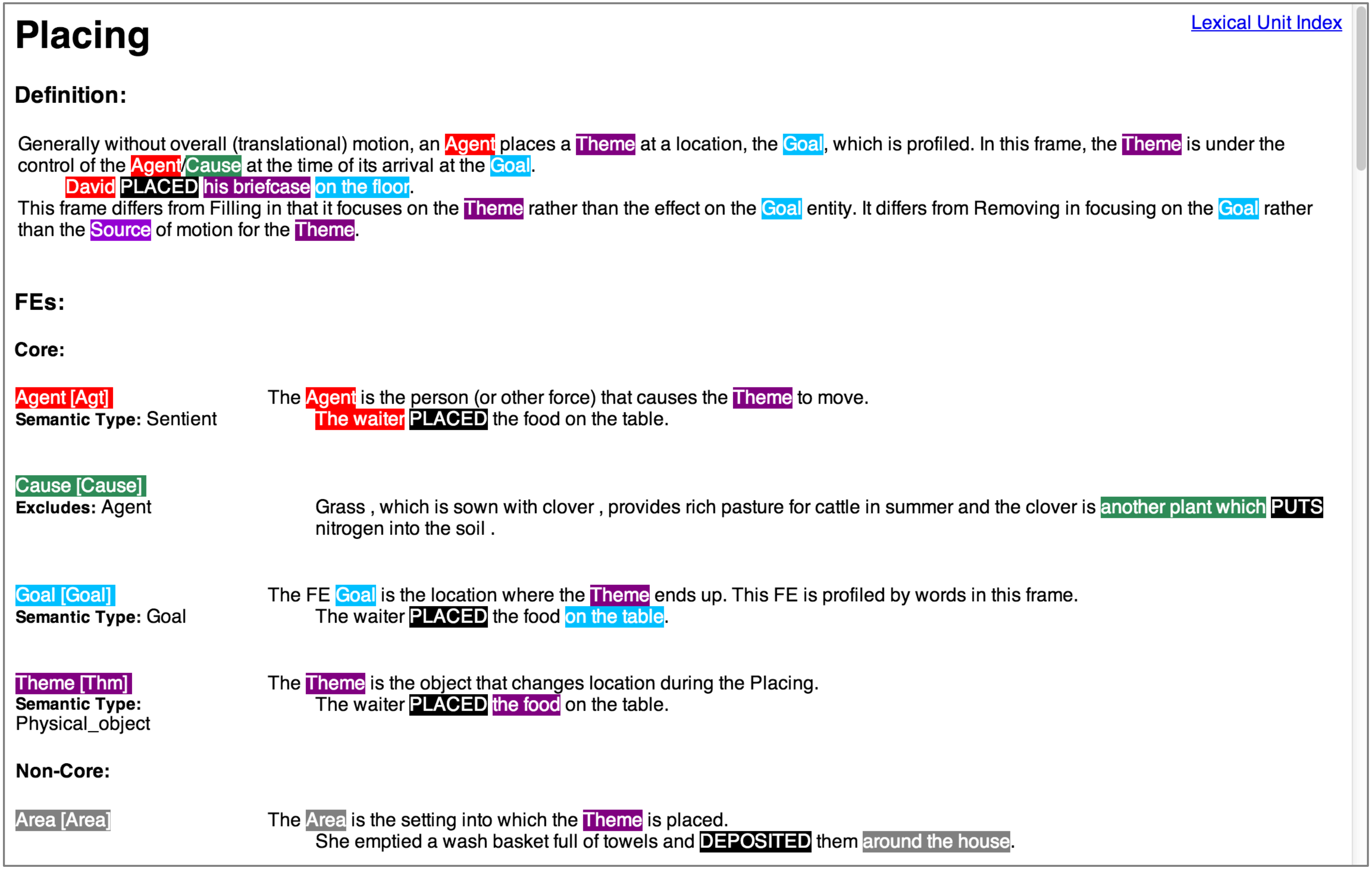
Grammatical Framework (GF)

- A toolbox for rapid development of multilingual CNLs
  - Provides a general-purpose resource grammar library (RGL) that encapsulates the low-level linguistic knowledge
  - All resource grammars implement a common **syntactic API**
  - Domain-specific, **semantic** application grammars (CNLs) are built on top of resource grammars
- Application grammar developers are **mapping** the semantic predicates to their syntactic constructors from scratch for each new/ported application grammar
  - Hypothesis: these mappings **can be reused** to a large extent providing a frame semantic abstraction layer to GF RGL

FrameNet (https://framenet.icsi.berkeley.edu)

- A semantic framework focused on **frame semantics**
  - Identifies >1000 **frames**: prototypical, language-independent situations with participating **frame elements** (semantic roles) – this can be seen as a **semantic 'API'**
    - We will focus on verb frames (~**600**) and their **core** elements
  - Identifies language-specific **lexical units** that evoke frames and their elements based on **syntactic valence patterns**
    - Mappings are derived from FrameNet-annotated **corpora** (being provided for an increasing number of languages)
- Limitation: FrameNet is not entirely formal and computational
  - There has been work on mapping FrameNet, for instance, to the formal SUMO ontology, or to other lexical resources like VerbNet and WordNet

Example frame



Placing

Lexical Unit Index

Definition:

Generally without overall (translational) motion, an Agent places a Theme at a location, the Goal, which is profiled. In this frame, the Theme is under the control of the Agent/Cause at the time of its arrival at the Goal.

David PLACED his briefcase on the floor.

This frame differs from Filling in that it focuses on the Theme rather than the effect on the Goal entity. It differs from Removing in focusing on the Goal rather than the Source of motion for the Theme.

FEs:

Core:

Agent [Agt]
Semantic Type: Sentient

The Agent is the person (or other force) that causes the Theme to move.
The waiter PLACED the food on the table.

Cause [Cause]
Excludes: Agent

Grass , which is sown with clover , provides rich pasture for cattle in summer and the clover is another plant which PUTS nitrogen into the soil .

Goal [Goal]
Semantic Type: Goal

The FE Goal is the location where the Theme ends up. This FE is profiled by words in this frame.
The waiter PLACED the food on the table.

Theme [Thm]
Semantic Type: Physical_object

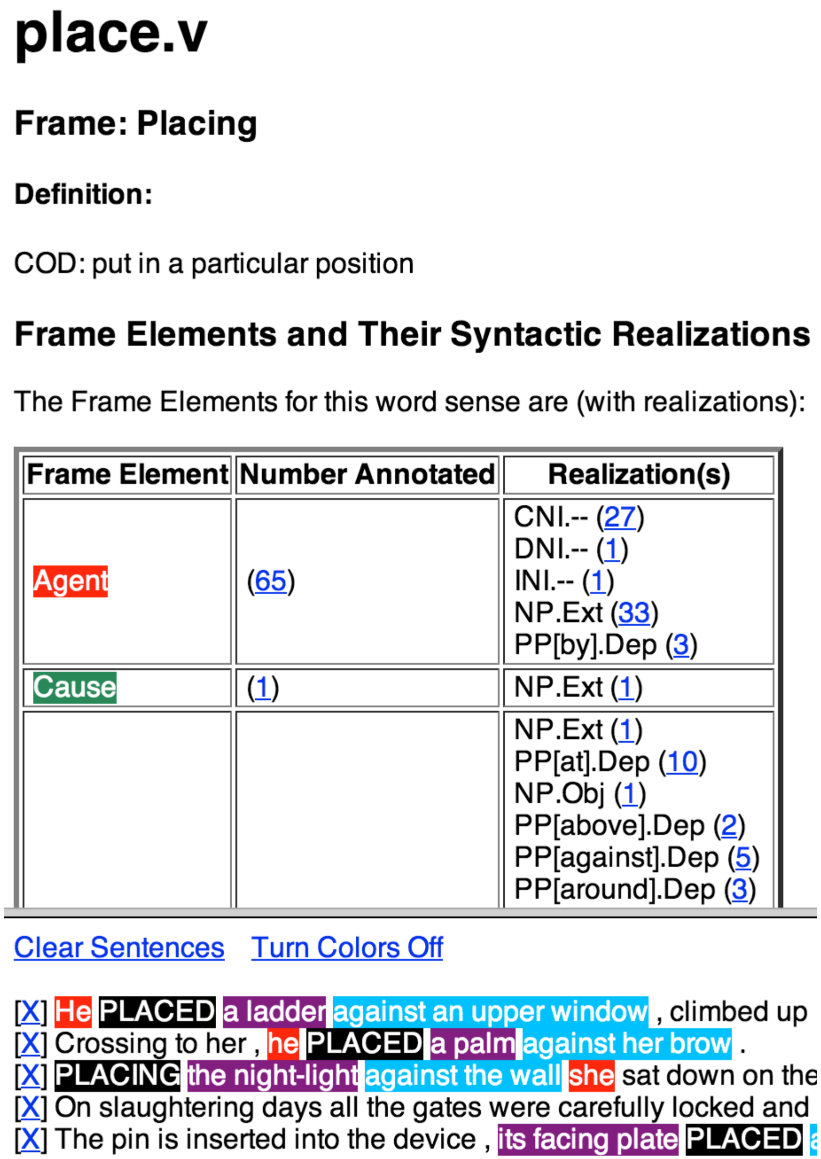
The Theme is the object that changes location during the Placing.
The waiter PLACED the food on the table.

Non-Core:

Area [Area]

The Area is the setting into which the Theme is placed.
She emptied a wash basket full of towels and DEPOSITED them around the house.

Example lexical entries



place.v

Frame: Placing

Definition:

COD: put in a particular position

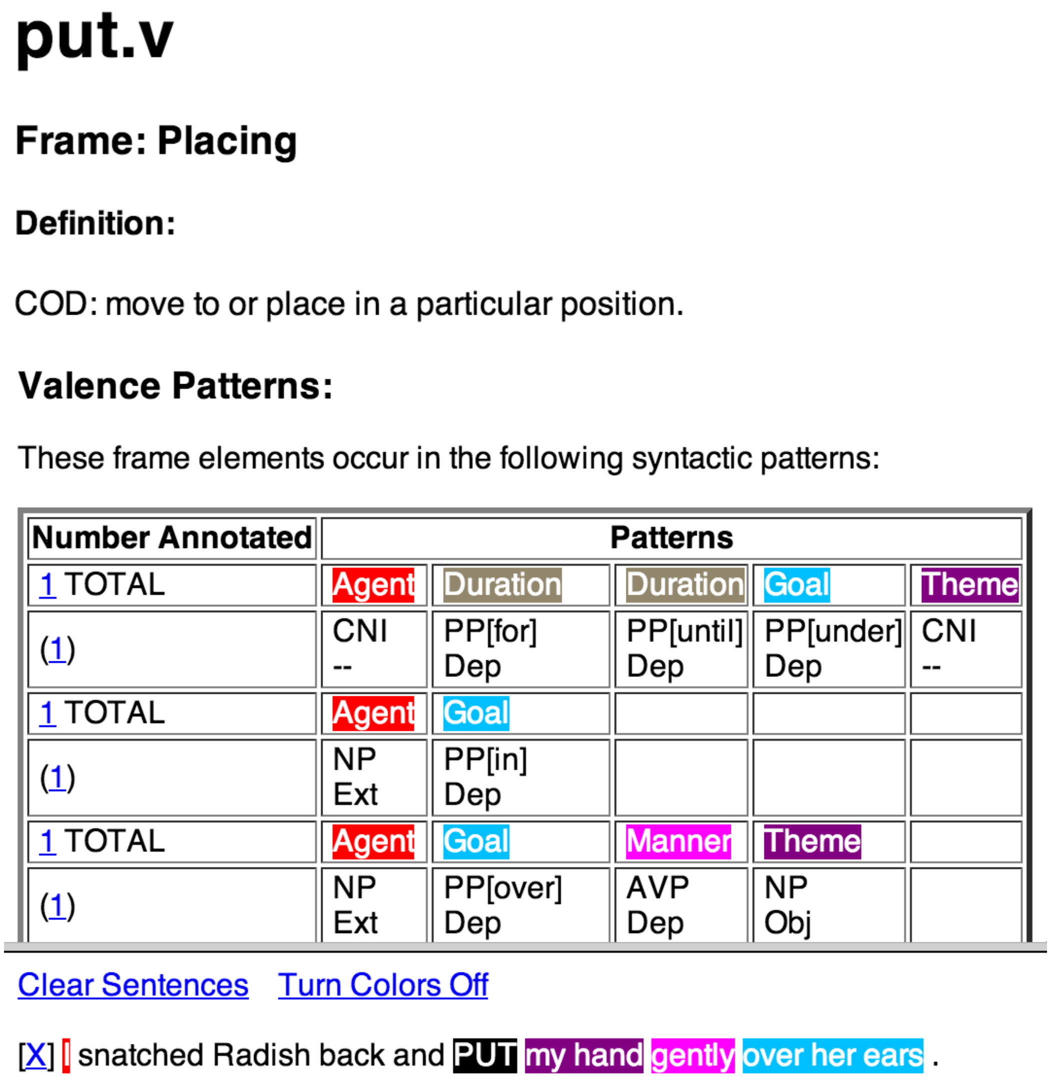
Frame Elements and Their Syntactic Realizations

The Frame Elements for this word sense are (with realizations):

Frame Element	Number Annotated	Realization(s)
Agent	(65)	CNI.-- (27) DNI.-- (1) INI.-- (1) NP.Ext (33) PP[by].Dep (3)
Cause	(1)	NP.Ext (1)
		NP.Ext (1) PP[at].Dep (10) NP.Obj (1) PP[above].Dep (2) PP[against].Dep (5) PP[around].Dep (3)

Clear Sentences Turn Colors Off

[X] He PLACED a ladder against an upper window , climbed up
[X] Crossing to her , he PLACED a palm against her brow .
[X] PLACING the night-light against the wall she sat down on the
[X] On slaughtering days all the gates were carefully locked and
[X] The pin is inserted into the device , its facing plate PLACED a



put.v

Frame: Placing

Definition:

COD: move to or place in a particular position.

Valence Patterns:

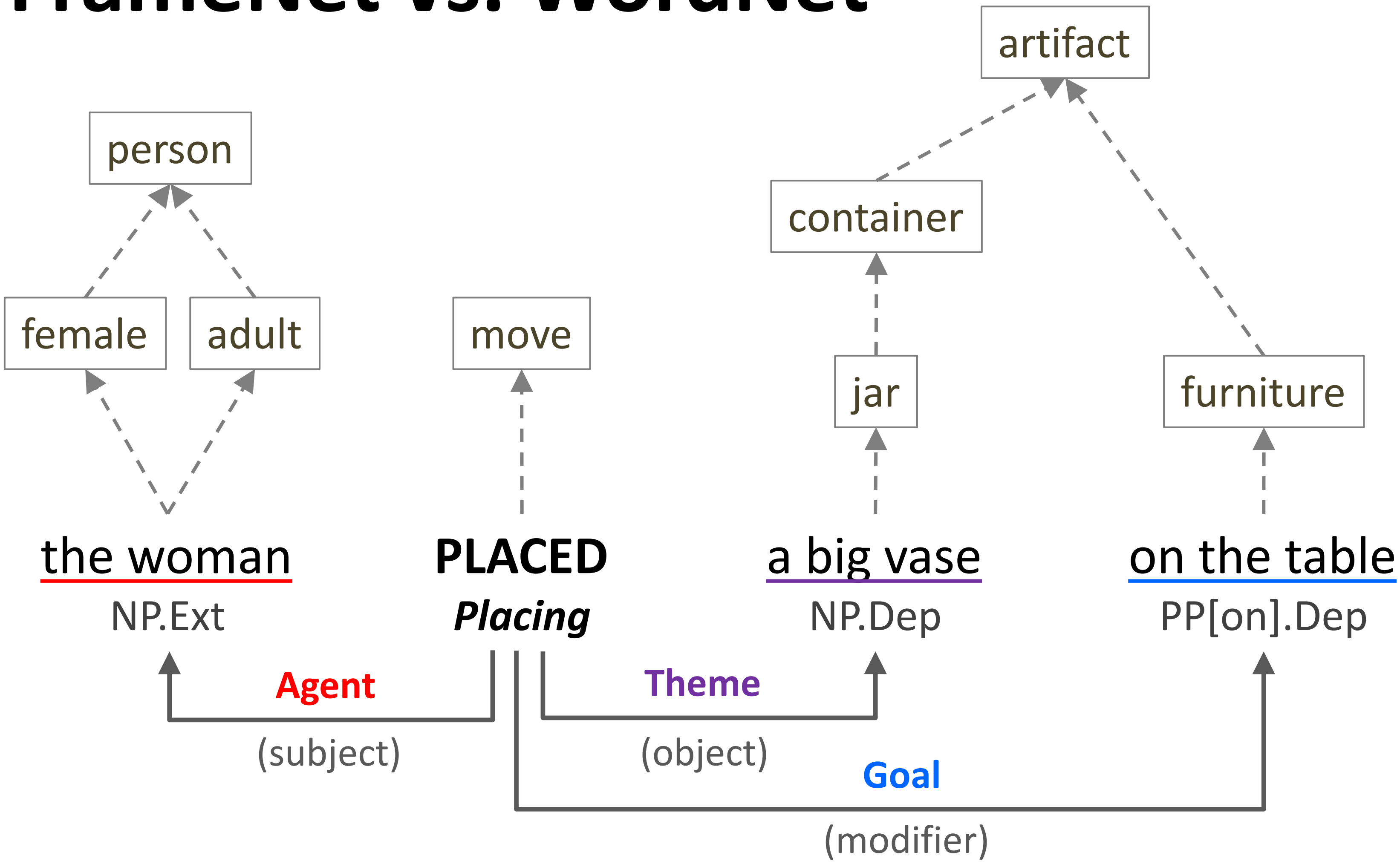
These frame elements occur in the following syntactic patterns:

Number Annotated	Patterns				
1 TOTAL	Agent	Duration	Duration	Goal	Theme
(1)	CNI --	PP[for] Dep	PP[until] Dep	PP[under] Dep	CNI --
1 TOTAL	Agent	Goal			
(1)	NP Ext	PP[in] Dep			
1 TOTAL	Agent	Goal	Manner	Theme	
(1)	NP Ext	PP[over] Dep	AVP Dep	NP Obj	

Clear Sentences Turn Colors Off

[X] I snatched Radish back and PUT my hand gently over her ears .

FrameNet vs. WordNet



vs. VerbNet: ~850 frame elements (FN) vs. ~25 general thematic roles (VN)

e.g., FN.***Being_employed***.Core: Employee, Employer, Field, Position, Task

Observations developing GF gramars

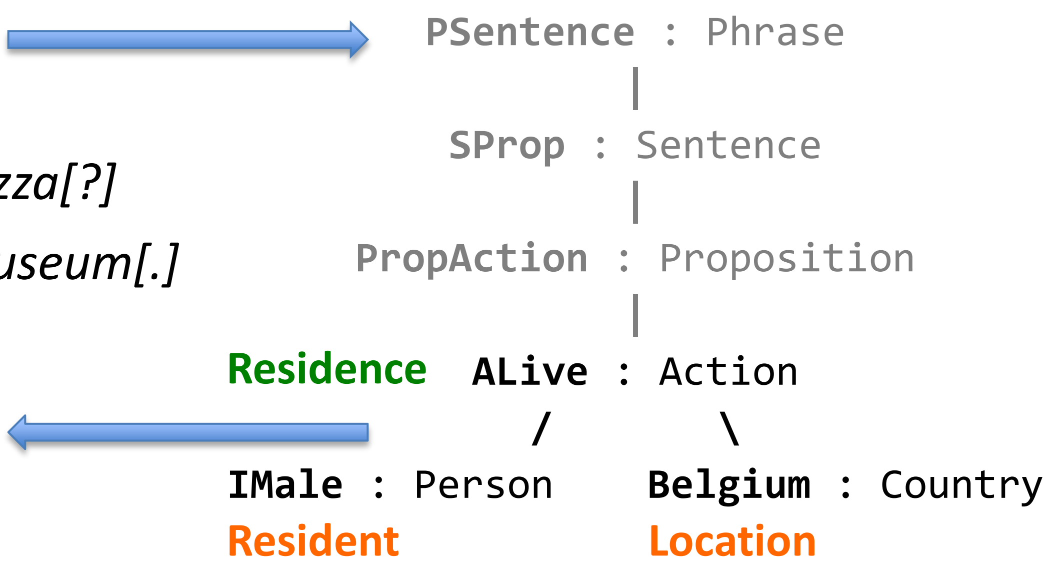
- When one gets used to..
  - the syntactic API
  - the typical syntactic patterns and trade-offs
- ..it becomes a rather routine work to "copy-paste-edit" the clause and VP level patterns
  - among different functions, languages, and even applications
  - providing a miniature domain-specific framenet for each application
- But beware of "exceptions": verb-dependent realizations of clauses (e.g. *love* vs. *like* in Russian, Italian, Latvian)
  - *Я*$_{\text{[NOM]}}$ ***люблю*** *тебя*$_{\text{[ACC]}}$ (*I love you*)
  - *Я*$_{\text{[NOM]}}$ ***нравлю*** *эту пиццу*$_{\text{[ACC]}}$ → *Мне*$_{\text{[DAT]}}$ ***нравится*** *эта пицца*$_{\text{[NOM]}}$
    (**I am pleasing this pizza** → *I like this pizza*)

Proposal: FrameNet API to RGL

- Building on top of GF RGL (but not extending it)
  - A common **semantic API**
  - Provides the **mapping** from the semantic frames and their core elements to their syntactic, language-dependent realization
- Application grammar (CNL) developers would manipulate with **semantic constructors**
  - Functions: the robust verb frames
  - Arguments: the core elements of the verb frames
    - From the syntactic view, they can be both arguments and adjuncts

Case-study: MOLTO Phrasebook

- Precise translation of standard touristic phrases
- Defines ~300 functions in the abstract syntax
  - – a lot of idiomatic phrases
  - – 20+ “actions” ≈ **frames** (`ALive`, `ALike`, `AWant`, `AWantGo` etc.)
    - *I live in Belgium[.]*
    - *I want a pizza[.]*
    - *[Do] you like this pizza[?]*
    - *I want to go to a museum[.]*



```
                    PSentence : Phrase
                              |
                       SProp : Sentence
                              |
                PropAction : Proposition
                              |
Residence            ALive : Action
                         /          \
IMale : Person            Belgium : Country
Resident                  Location
```

Abito in Belgio[.]
(Italian)

Phrasebook: English

```
Belgium = mkNP (mkPN "Belgium") ;
Museum = mkPlaceKind "museum" "at" ;
Pizza = mkCN (mkN "pizza") ;
```

```
-- Cl -> NP VP // VP -> VP Adv // Adv -> Prep NP
ALive pers country = mkCl pers.name
  (mkVP (mkVP (mkV "live")) (mkAdv SyntaxEng.in_Prep country)) ;

-- Cl -> NP V2 NP
ALike pers item = mkCl pers.name (mkV2 (mkV "like")) item ;

-- Cl -> NP V2 NP
AWant pers obj = mkCl pers.name (mkV2 (mkV "want")) obj ;

-- Cl -> NP VV VP // VP -> VP Adv
AWantGo pers place = mkCl pers.name SyntaxEng.want_VV
  (mkVP (mkVP IrregEng.go_V) place.to) ;
```

Semantic vs. syntactic constructors

- ALive *p co* =

```
    Residence live_V  p.name    NIL          co
                      Resident  Co_resident  Location
```

- ALike *p it* =

```
    Experiencer_focus like_V  p.name       it       NIL
                              Experiencer  Content  Topic
```

- AWantGo *p pl* =

```
    Desiring want_V  p.name       (Motion go_V NIL     pl.name)
                     Experiencer  Event            Source  Goal
```

Function	Arguments	Value
Residence	V Resident Location Co_resident	Cl
Experiencer_focus	V Experiencer Content Topic	Cl
Motion	V Theme Source Goal	Cl
Motion	V Source Goal	VP

Statistics from a FrameNet corpus

- E.g. the lexical entry `Residence.live`:

Core FE	Total	Pattern
Resident	143	NP.Ext (90%) xNI.-- (9%)
Co_resident	14	PP.Dep (86%) xNI.– (14%)
Location	131	PP.Dep (81%) AVP.Dep (13%)

with	9	=/=	*in*	72
among	3		*on*	8
			at	4
			…	

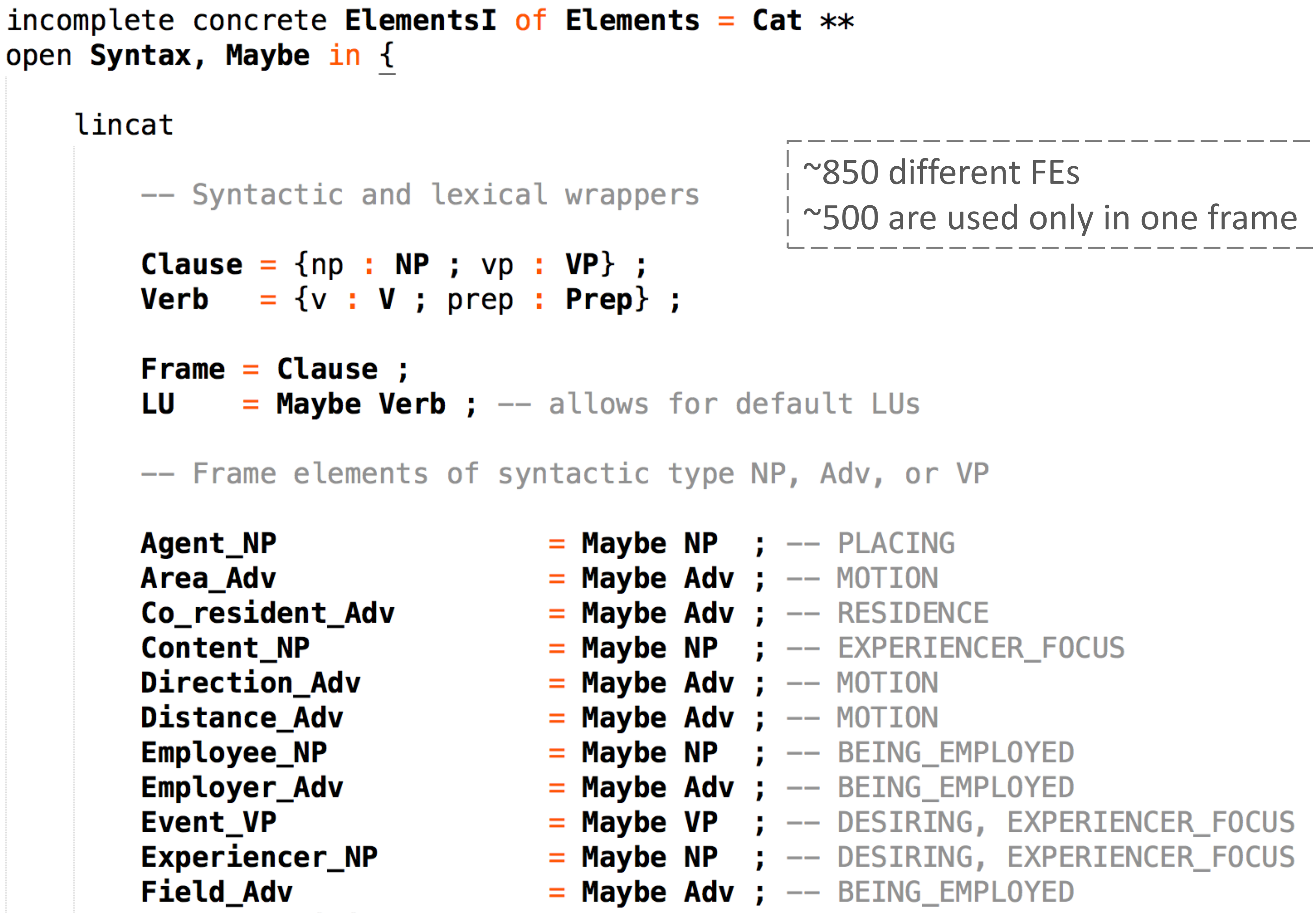
Total	Patterns		
98	**Resident**		**Location**
71%	NP.Ext		PP.Dep
17%	NP.Ext		AVP
7	**Resident**	**Co_resident**	
86%	NP.Ext	PP.Dep	
7	**Resident**	**Co_resident**	**Location**
86%	NP.Ext	PP.Dep	PP.Dep
112			
79%			

P.S. In GF, Adv includes PP

Assumptions

- For every combination of FE types, there is a common syntactic realization of a frame that is reused by **most verbs**
  - There can be **different agreement** patterns that are specific to particular verbs or groups of verbs (systematic exceptions)
  - Prepositions, in general, do not depend on the frame, although often there is a **dominant preposition** per frame element (if realized as a PP)
- In the CNL settings, it is often sufficient that **only core elements** (according to FrameNet) are available
- It is possible to choose a **default lexical unit** per frame to be used in the linearization, if a specific verb is not provided
  - The most general and/or the most frequently used LU

Prototype #1: frame elements

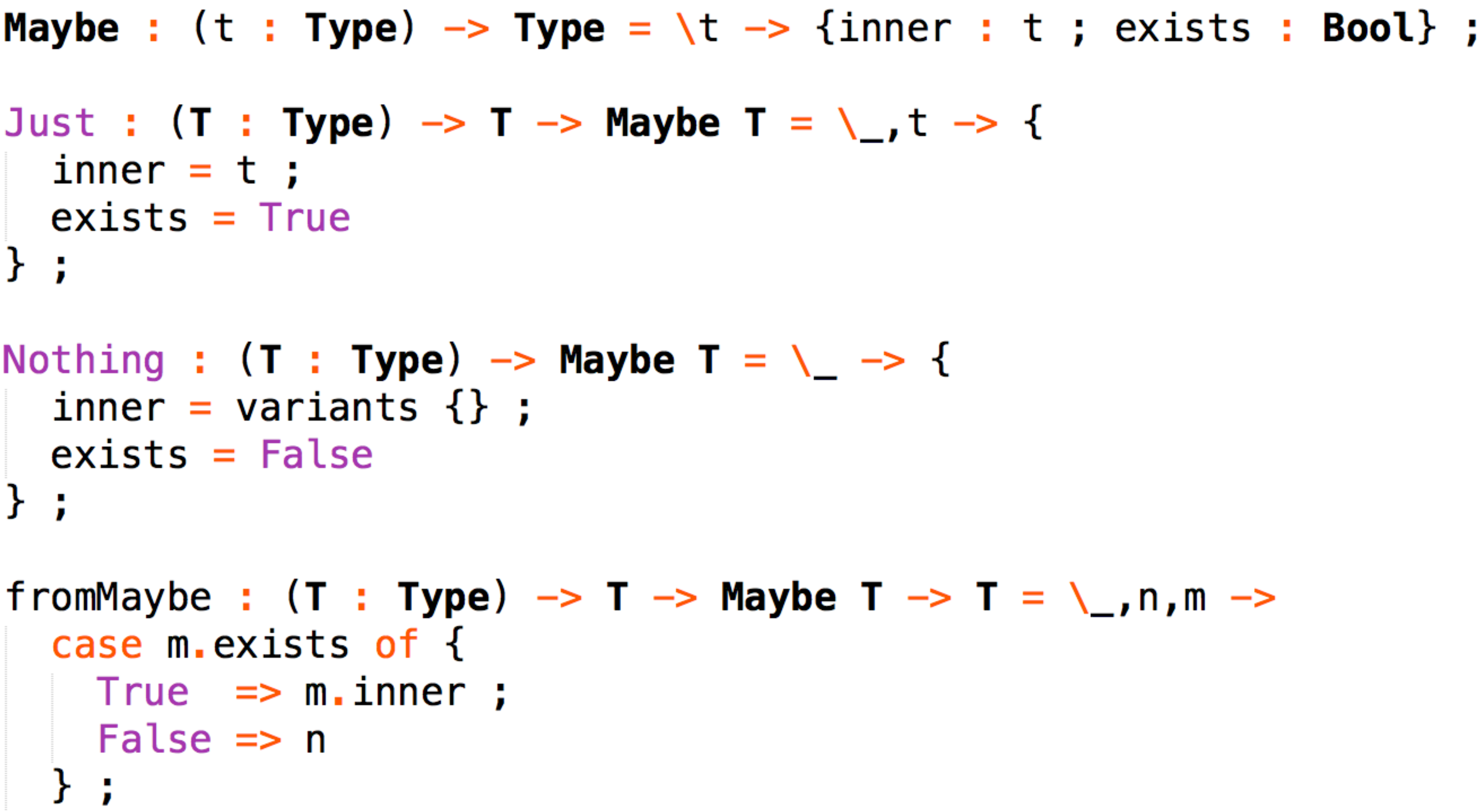


```
incomplete concrete ElementsI of Elements = Cat **
open Syntax, Maybe in {

    lincat

        -- Syntactic and lexical wrappers

        Clause = {np : NP ; vp : VP} ;
        Verb   = {v : V ; prep : Prep} ;

        Frame = Clause ;
        LU    = Maybe Verb ; -- allows for default LUs

        -- Frame elements of syntactic type NP, Adv, or VP

        Agent_NP                = Maybe NP  ; -- PLACING
        Area_Adv                = Maybe Adv ; -- MOTION
        Co_resident_Adv         = Maybe Adv ; -- RESIDENCE
        Content_NP              = Maybe NP  ; -- EXPERIENCER_FOCUS
        Direction_Adv           = Maybe Adv ; -- MOTION
        Distance_Adv            = Maybe Adv ; -- MOTION
        Employee_NP             = Maybe NP  ; -- BEING_EMPLOYED
        Employer_Adv            = Maybe Adv ; -- BEING_EMPLOYED
        Event_VP                = Maybe VP  ; -- DESIRING, EXPERIENCER_FOCUS
        Experiencer_NP          = Maybe NP  ; -- DESIRING, EXPERIENCER_FOCUS
        Field_Adv               = Maybe Adv ; -- BEING_EMPLOYED
```

~850 different FEs

~500 are used only in one frame

The Maybe type

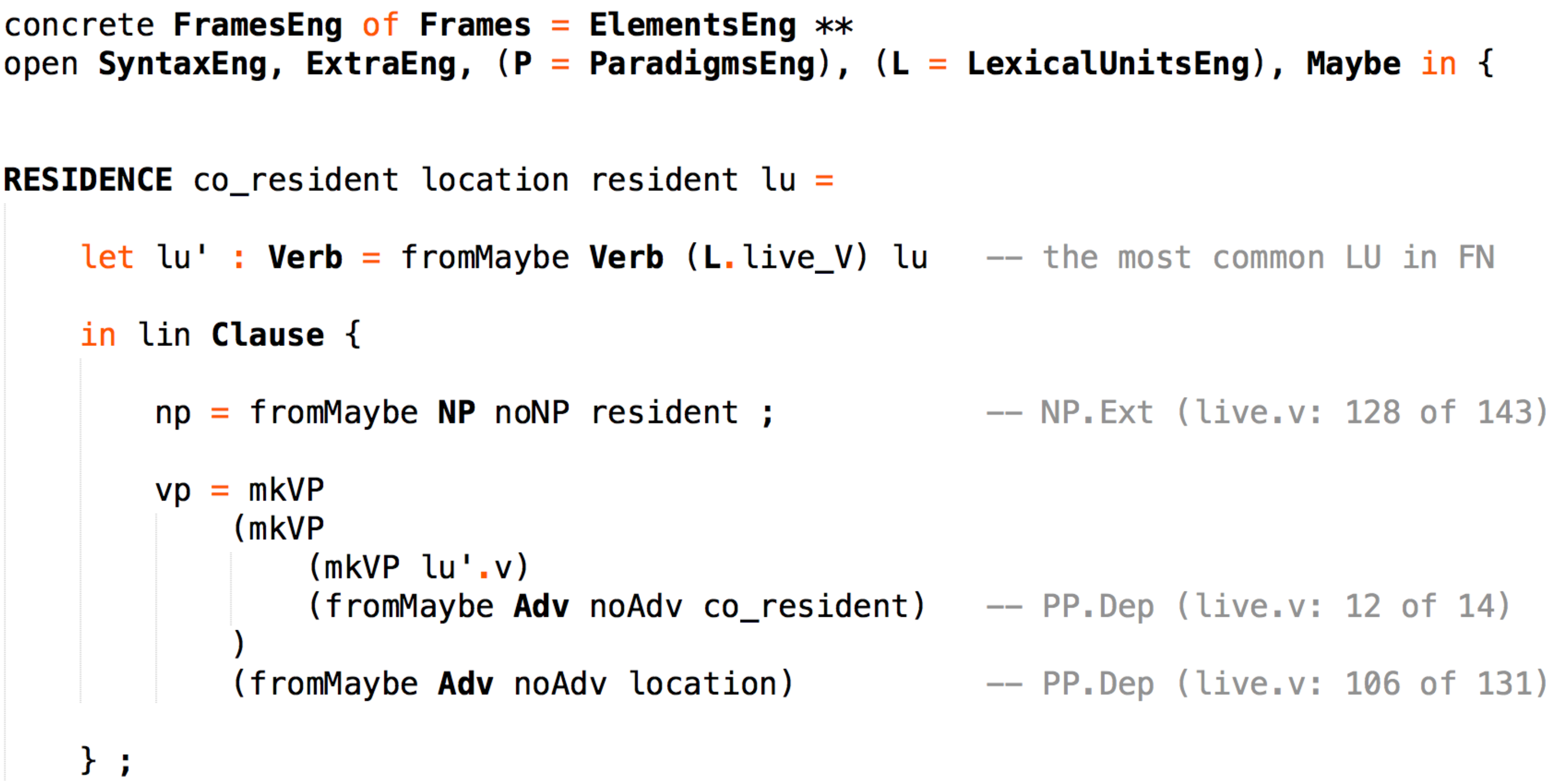


```
Maybe : (t : Type) -> Type = \t -> {inner : t ; exists : Bool} ;

Just : (T : Type) -> T -> Maybe T = \_,t -> {
  inner = t ;
  exists = True
} ;

Nothing : (T : Type) -> Maybe T = \_ -> {
  inner = variants {} ;
  exists = False
} ;

fromMaybe : (T : Type) -> T -> Maybe T -> T = \_,n,m ->
  case m.exists of {
    True  => m.inner ;
    False => n
  } ;
```

Prototype #1: frames (abstract syntax)

```
abstract Frames = Elements ** {

-- RESIDENCE: This frame has to do with people (the Residents) residing in Locations...
--
-- Co_resident: A person or group of people that the resident is staying with or among.
-- Location   : The place in which somebody resides.
-- Resident   : The individual(s) that reside at the Location.
RESIDENCE : Co_resident_Adv -> Location_Adv -> Resident_NP -> LU -> Frame ;

-- PLACING: Generally without overall (translational) motion, an Agent places a Theme...
-- ...
-- Cause excludes Agent. (Only one questionable example in the corpus; similar to Agent.)
PLACING : Agent_NP -> Goal_Adv -> Theme_NP -> LU -> Frame ;

-- DESIRING: An Experiencer desires that an Event occur...
--
-- Event            : The change that the Experiencer would like to see.
-- Experiencer      : ...
-- Focal_participant: The entity that the Experiencer wishes to be affected by some Event.
-- Location_of_Event: The Location_of_Event is the place involved in the desired Event.
--
-- Event and Focal_participant are (*actually*) mutually excluding.
-- Location_of_Event is (*actually*) non-core.
DESIRING_Event             : Event_VP -> Experiencer_NP ->                         LU -> Frame ;
DESIRING_Focal_participant :             Experiencer_NP -> Focal_participant_NP -> LU -> Frame ;

-- BEING_EMPLOYED: ...
BEING_EMPLOYED_Task_Adv : Employee_NP -> Employer_Adv -> ... -> Task_Adv -> LU -> Frame ;
BEING_EMPLOYED_Task_VP  : Employee_NP -> Employer_Adv -> ... -> Task_VP  -> LU -> Frame ;
```

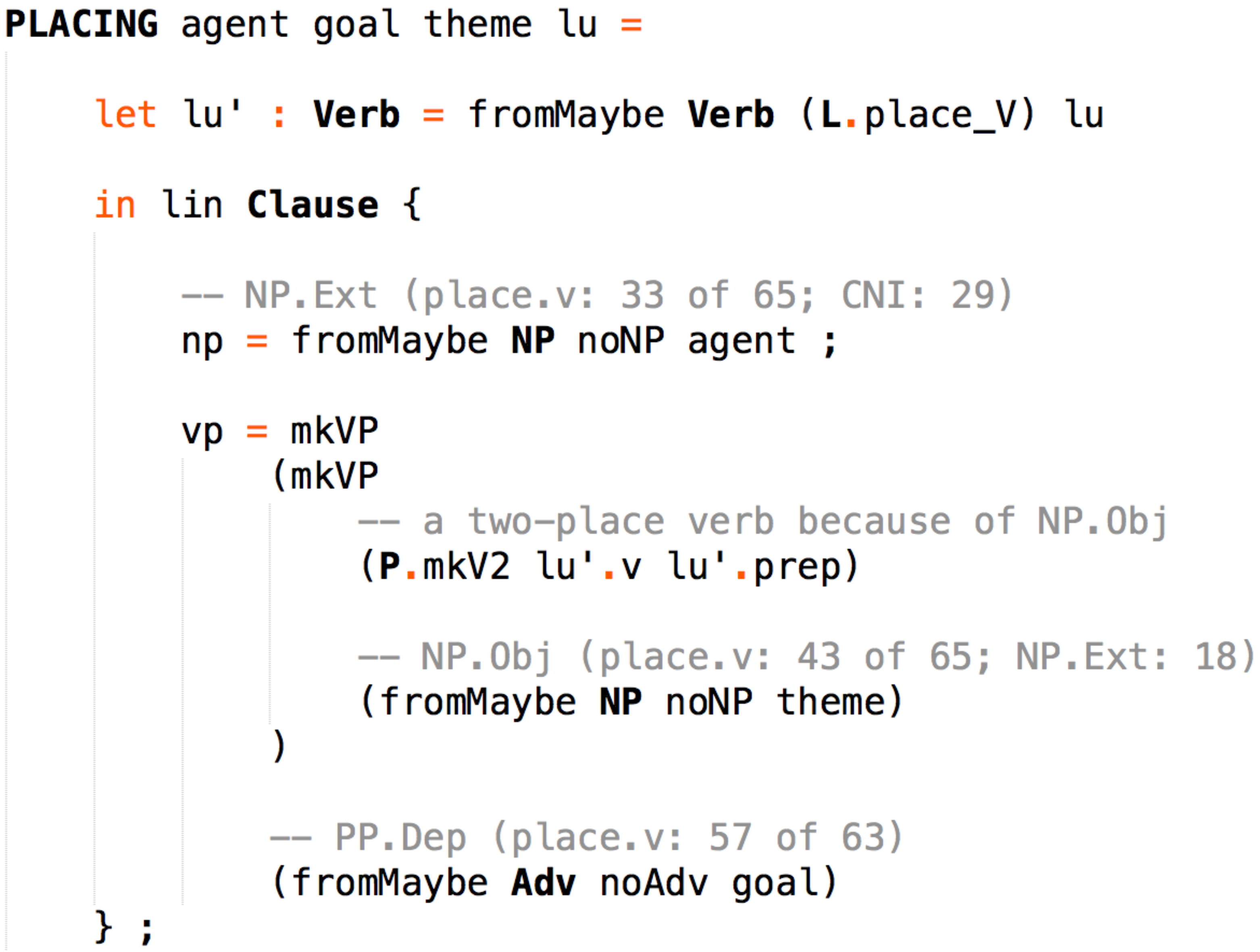
Prototype #1: frames in English



```
concrete FramesEng of Frames = ElementsEng **
open SyntaxEng, ExtraEng, (P = ParadigmsEng), (L = LexicalUnitsEng), Maybe in {


RESIDENCE co_resident location resident lu =

    let lu' : Verb = fromMaybe Verb (L.live_V) lu  -- the most common LU in FN

    in lin Clause {

        np = fromMaybe NP noNP resident ;           -- NP.Ext (live.v: 128 of 143)

        vp = mkVP
            (mkVP
                (mkVP lu'.v)
                (fromMaybe Adv noAdv co_resident)   -- PP.Dep (live.v: 12 of 14)
            )
            (fromMaybe Adv noAdv location)          -- PP.Dep (live.v: 106 of 131)

    } ;
```

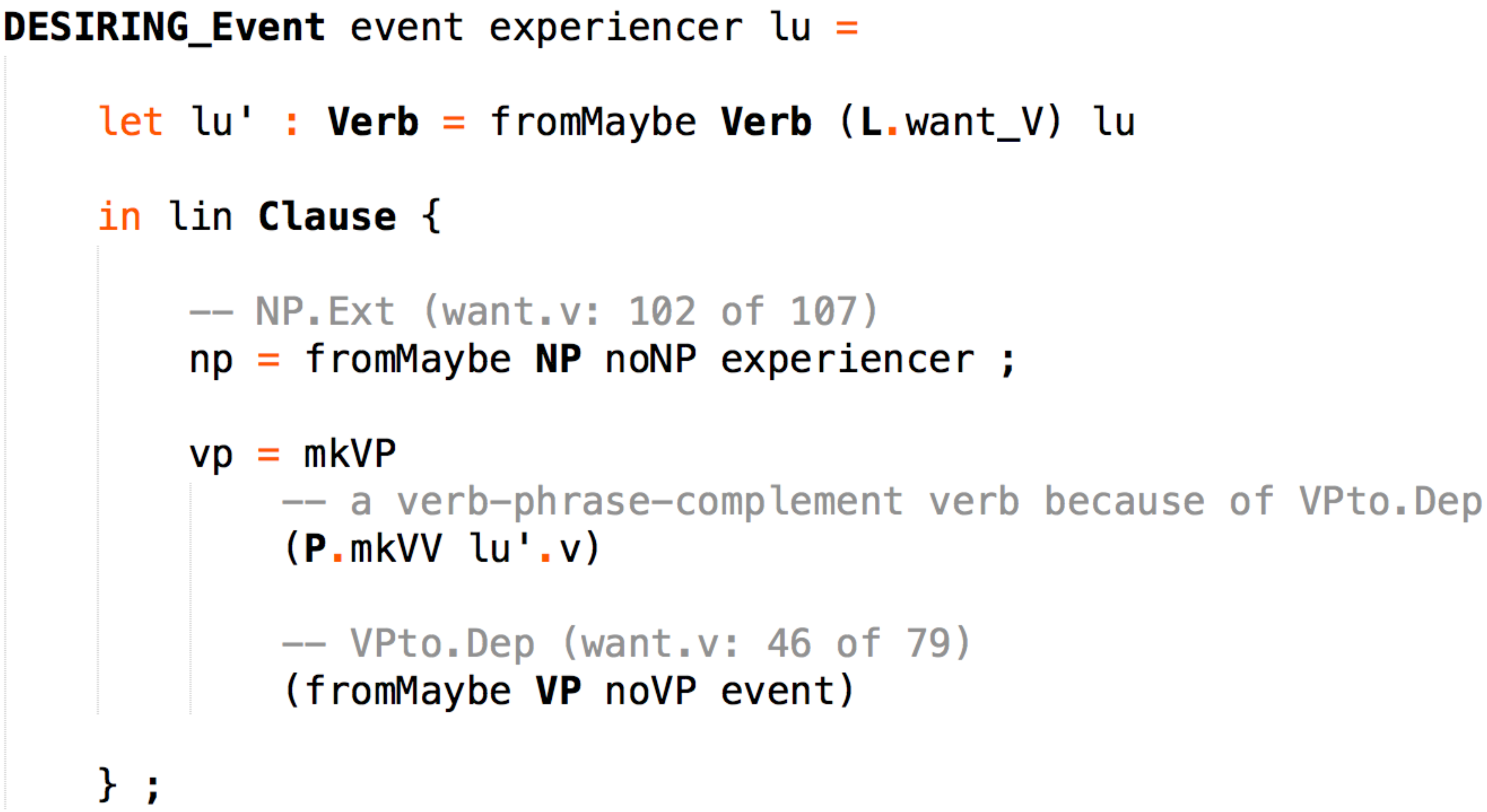
Side effect: all core elements (= essential to the meaning of a frame) appear in AST even if they are not directly expressed in the sentence (P.S. Well, currently no FEs will appear...)

Prototype #1: frames in English



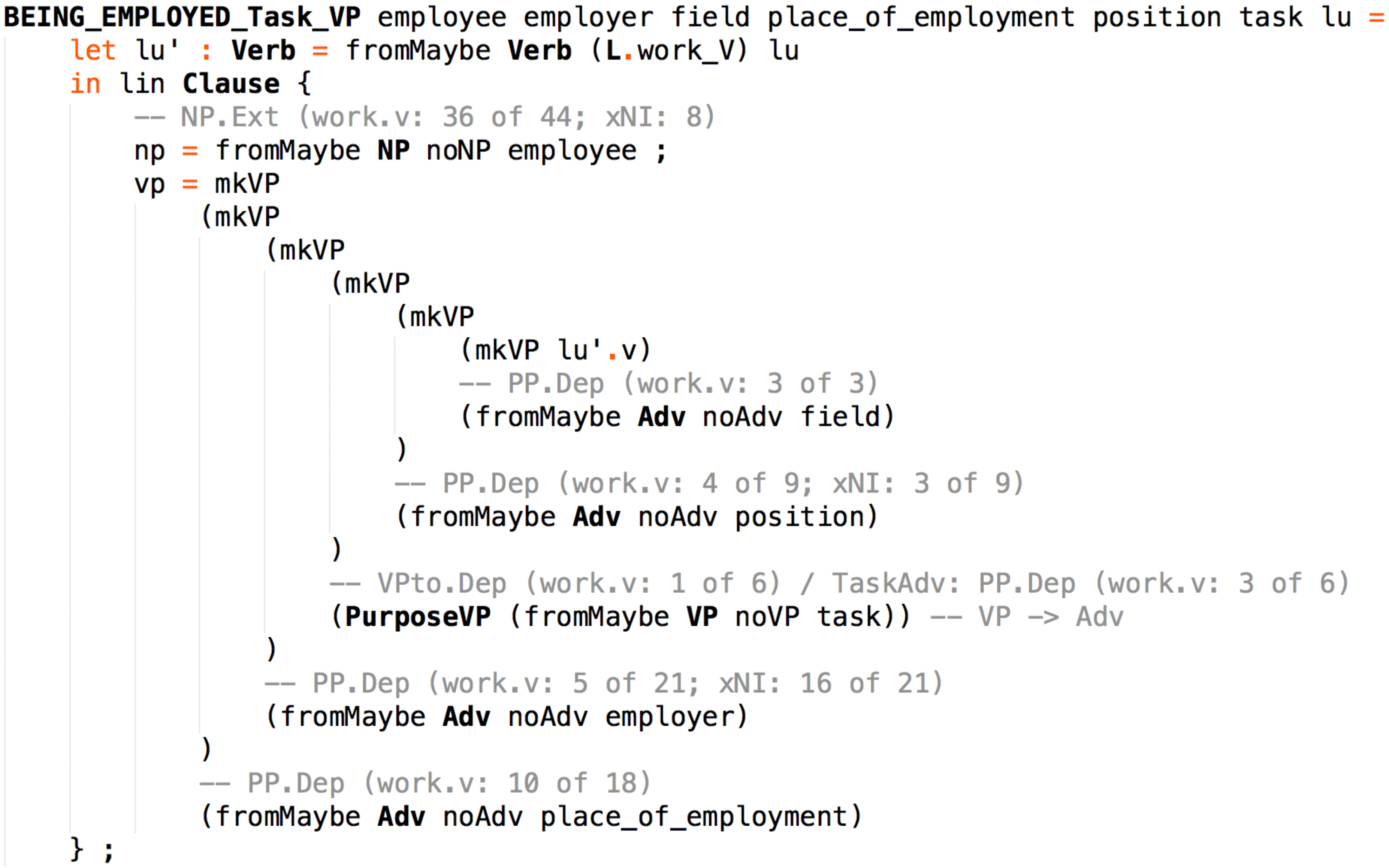
```
PLACING agent goal theme lu =
    let lu' : Verb = fromMaybe Verb (L.place_V) lu
    in lin Clause {
        -- NP.Ext (place.v: 33 of 65; CNI: 29)
        np = fromMaybe NP noNP agent ;

        vp = mkVP
            (mkVP
                -- a two-place verb because of NP.Obj
                (P.mkV2 lu'.v lu'.prep)

                -- NP.Obj (place.v: 43 of 65; NP.Ext: 18)
                (fromMaybe NP noNP theme)
            )

            -- PP.Dep (place.v: 57 of 63)
            (fromMaybe Adv noAdv goal)
    } ;
```

Prototype #1: frames in English



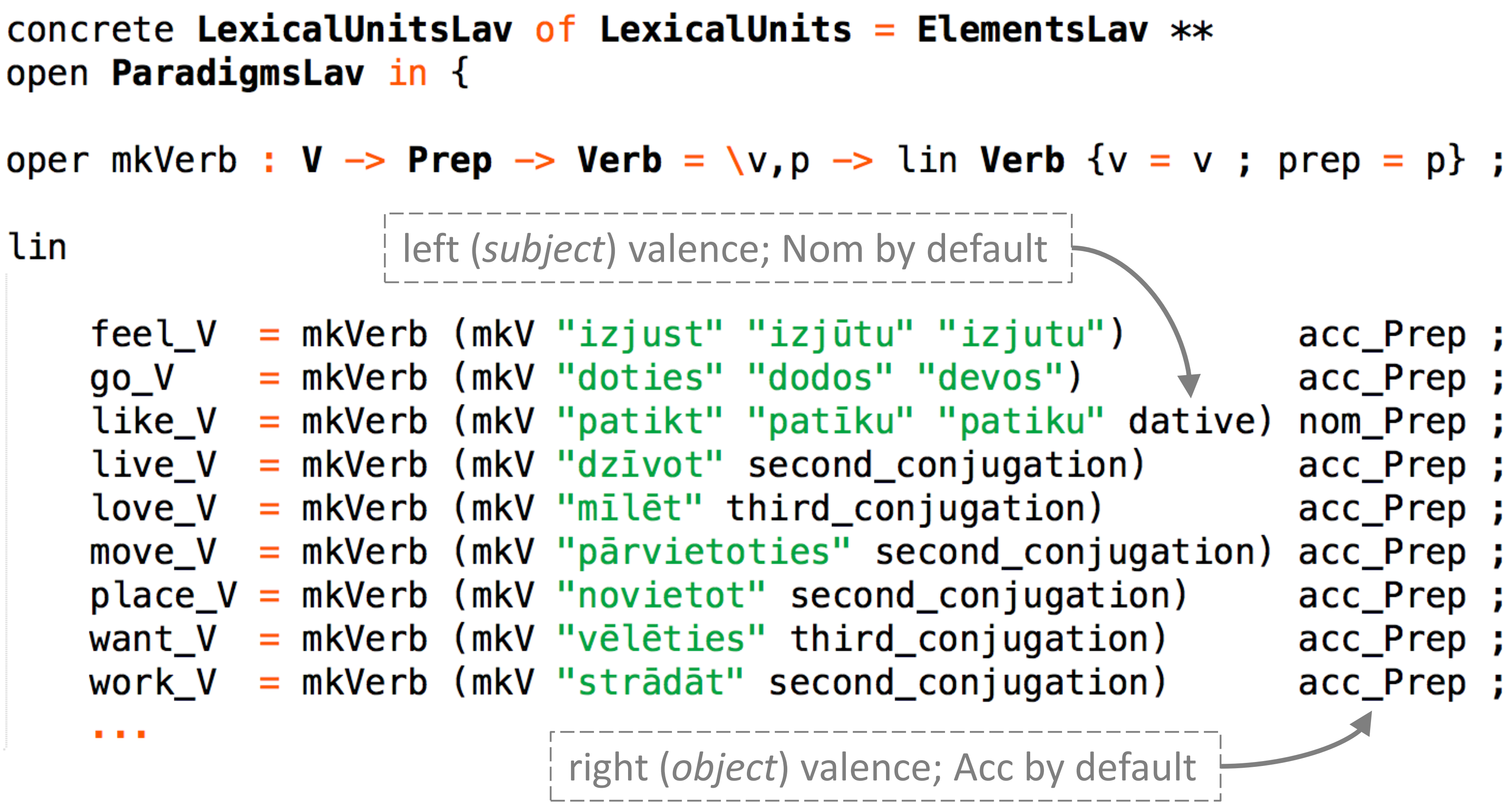
```
DESIRING_Event event experiencer lu =
    let lu' : Verb = fromMaybe Verb (L.want_V) lu
    in lin Clause {
        -- NP.Ext (want.v: 102 of 107)
        np = fromMaybe NP noNP experiencer ;
        vp = mkVP
            -- a verb-phrase-complement verb because of VPto.Dep
            (P.mkVV lu'.v)

            -- VPto.Dep (want.v: 46 of 79)
            (fromMaybe VP noVP event)
    } ;
```

Prototype #1: frames in English



```
BEING_EMPLOYED_Task_VP employee employer field place_of_employment position task lu =
    let lu' : Verb = fromMaybe Verb (L.work_V) lu
    in lin Clause {
        -- NP.Ext (work.v: 36 of 44; xNI: 8)
        np = fromMaybe NP noNP employee ;
        vp = mkVP
            (mkVP
                (mkVP
                    (mkVP
                        (mkVP
                            (mkVP lu'.v)
                            -- PP.Dep (work.v: 3 of 3)
                            (fromMaybe Adv noAdv field)
                        )
                        -- PP.Dep (work.v: 4 of 9; xNI: 3 of 9)
                        (fromMaybe Adv noAdv position)
                    )
                    -- VPto.Dep (work.v: 1 of 6) / TaskAdv: PP.Dep (work.v: 3 of 6)
                    (PurposeVP (fromMaybe VP noVP task)) -- VP -> Adv
                )
                -- PP.Dep (work.v: 5 of 21; xNI: 16 of 21)
                (fromMaybe Adv noAdv employer)
            )
            -- PP.Dep (work.v: 10 of 18)
            (fromMaybe Adv noAdv place_of_employment)
    } ;
```

Prototype #1: frames in Latvian



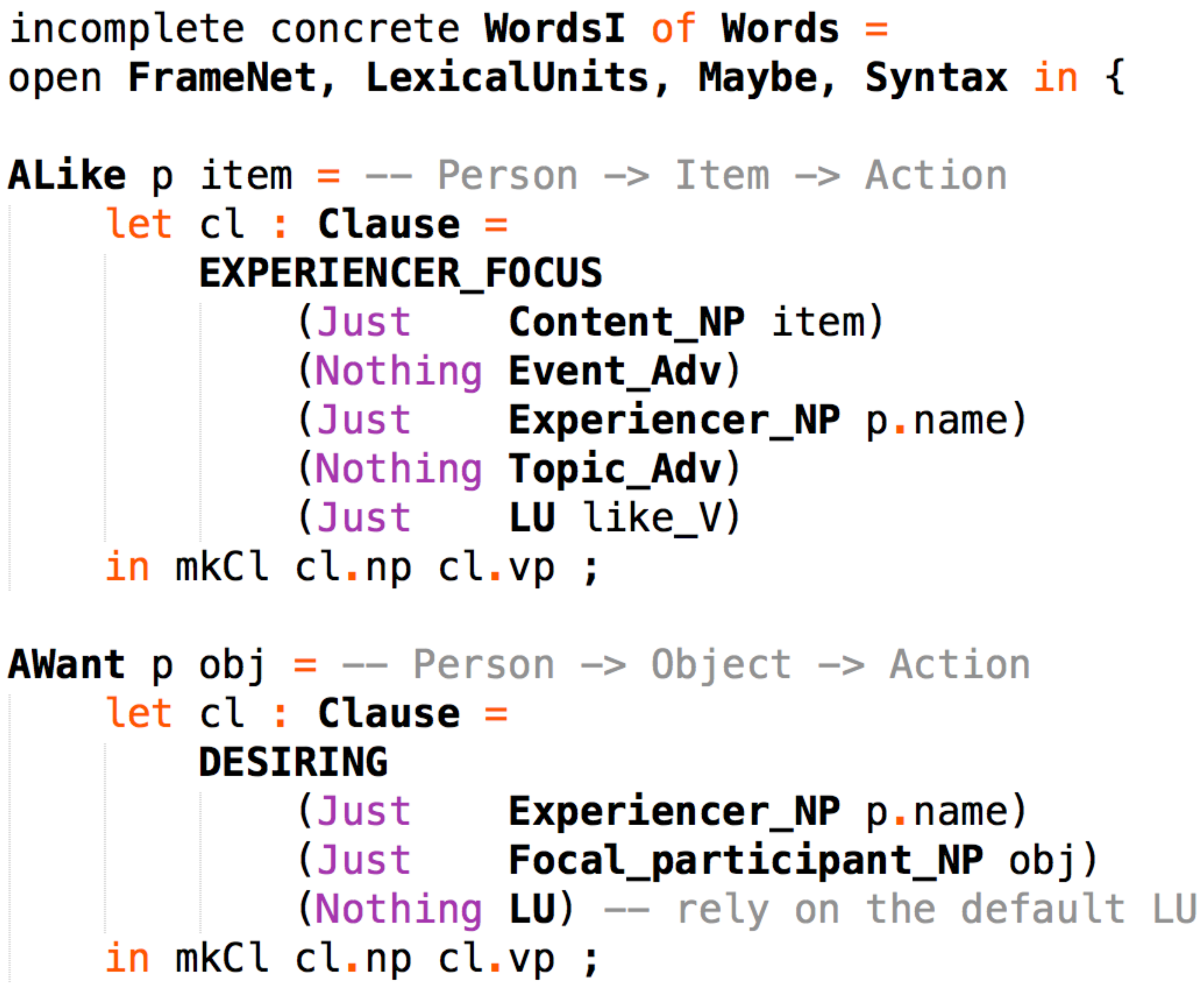
```
concrete LexicalUnitsLav of LexicalUnits = ElementsLav **
open ParadigmsLav in {

oper mkVerb : V -> Prep -> Verb = \v,p -> lin Verb {v = v ; prep = p} ;

lin

    feel_V  = mkVerb (mkV "izjust" "izjūtu" "izjutu")        acc_Prep ;
    go_V    = mkVerb (mkV "doties" "dodos" "devos")          acc_Prep ;
    like_V  = mkVerb (mkV "patikt" "patīku" "patiku" dative) nom_Prep ;
    live_V  = mkVerb (mkV "dzīvot" second_conjugation)       acc_Prep ;
    love_V  = mkVerb (mkV "mīlēt" third_conjugation)         acc_Prep ;
    move_V  = mkVerb (mkV "pārvietoties" second_conjugation) acc_Prep ;
    place_V = mkVerb (mkV "novietot" second_conjugation)     acc_Prep ;
    want_V  = mkVerb (mkV "vēlēties" third_conjugation)      acc_Prep ;
    work_V  = mkVerb (mkV "strādāt" second_conjugation)      acc_Prep ;
    ...
```

left (*subject*) valence; Nom by default

right (*object*) valence; Acc by default

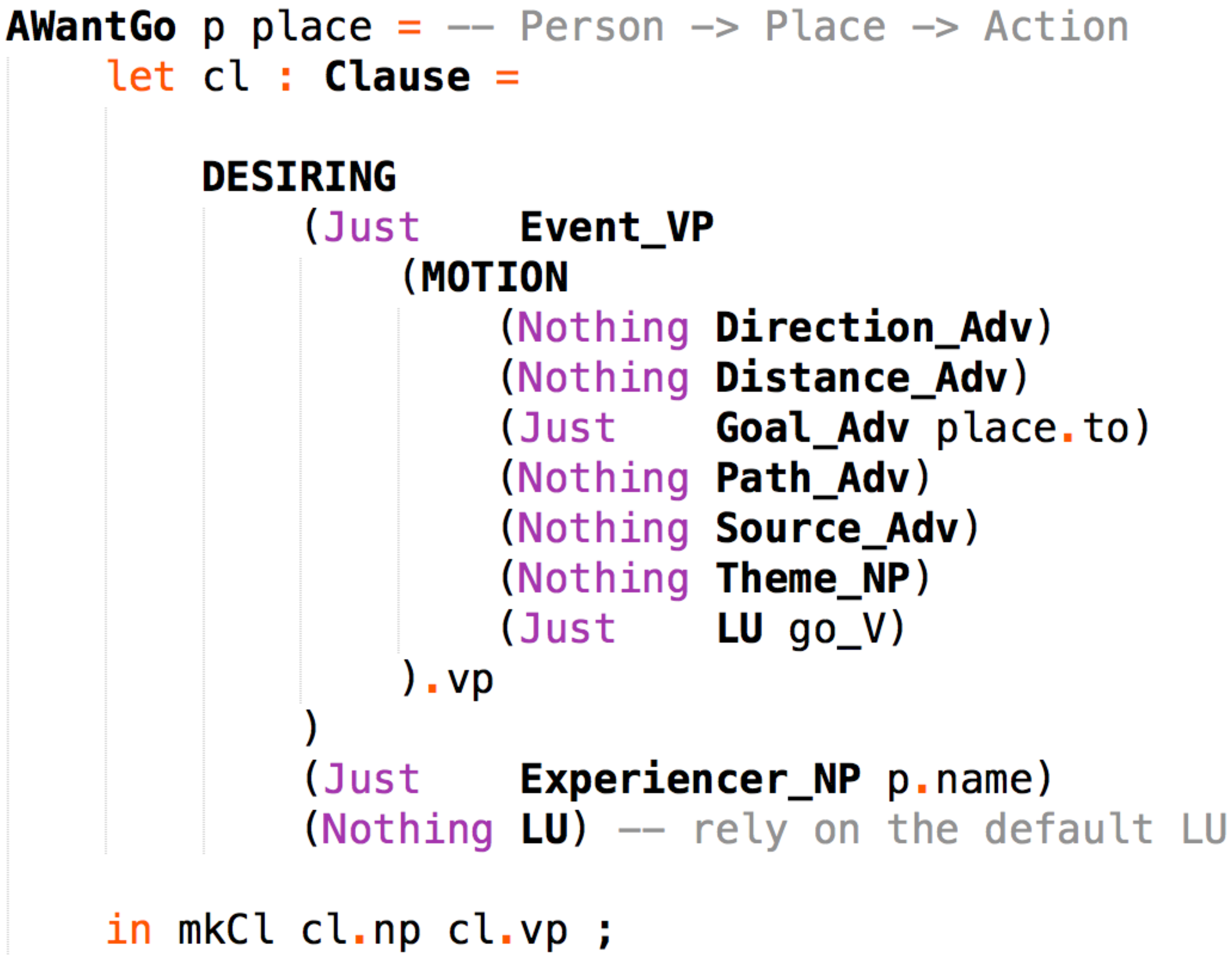
Otherwise, at this level of FE abstraction, **copy-paste** from English! Thus, a **functor** for frames should be possible...

Usage: in a Phrasebook functor



```
incomplete concrete WordsI of Words =
open FrameNet, LexicalUnits, Maybe, Syntax in {

ALike p item = -- Person -> Item -> Action
    let cl : Clause =
        EXPERIENCER_FOCUS
            (Just    Content_NP item)
            (Nothing Event_Adv)
            (Just    Experiencer_NP p.name)
            (Nothing Topic_Adv)
            (Just    LU like_V)
    in mkCl cl.np cl.vp ;

AWant p obj = -- Person -> Object -> Action
    let cl : Clause =
        DESIRING
            (Just    Experiencer_NP p.name)
            (Just    Focal_participant_NP obj)
            (Nothing LU) -- rely on the default LU
    in mkCl cl.np cl.vp ;
```

Usage: in a Phrasebook functor



```
AWantGo p place = -- Person -> Place -> Action
    let cl : Clause =

        DESIRING
            (Just    Event_VP
                (MOTION
                    (Nothing Direction_Adv)
                    (Nothing Distance_Adv)
                    (Just    Goal_Adv place.to)
                    (Nothing Path_Adv)
                    (Nothing Source_Adv)
                    (Nothing Theme_NP)
                    (Just    LU go_V)
                ).vp
            )
            (Just    Experiencer_NP p.name)
            (Nothing LU) -- rely on the default LU

    in mkCl cl.np cl.vp ;
```

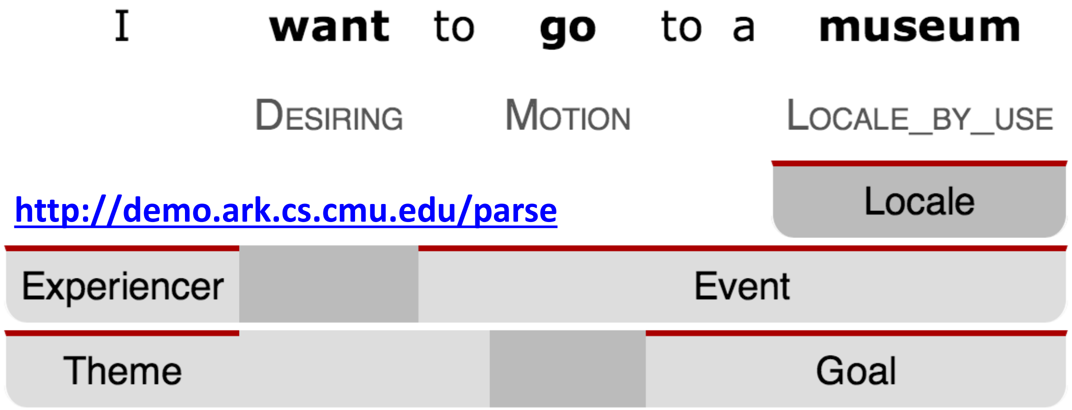
Usage: as a general application grammar

- At this level of syntactic abstraction of frame elements..
  - .. do we really need the full FrameNet just to facilitate the development of certain kind of application grammars?
  - Many frames are implemented in the same way as some other frames
    - A smaller set of more general (more syntactic) frames might be sufficient to achieve the same effect
- The FrameNet resource library could be used on its own:
  - for semantic parsing
  - for natural language generation (from FrameNet-annotated data)

```
               DESIRING
              /   |   \
    Experiencer   T    Event
         |        |      |
      i_Pron   want_V  MOTION
                       /     \
                      T      Goal
```

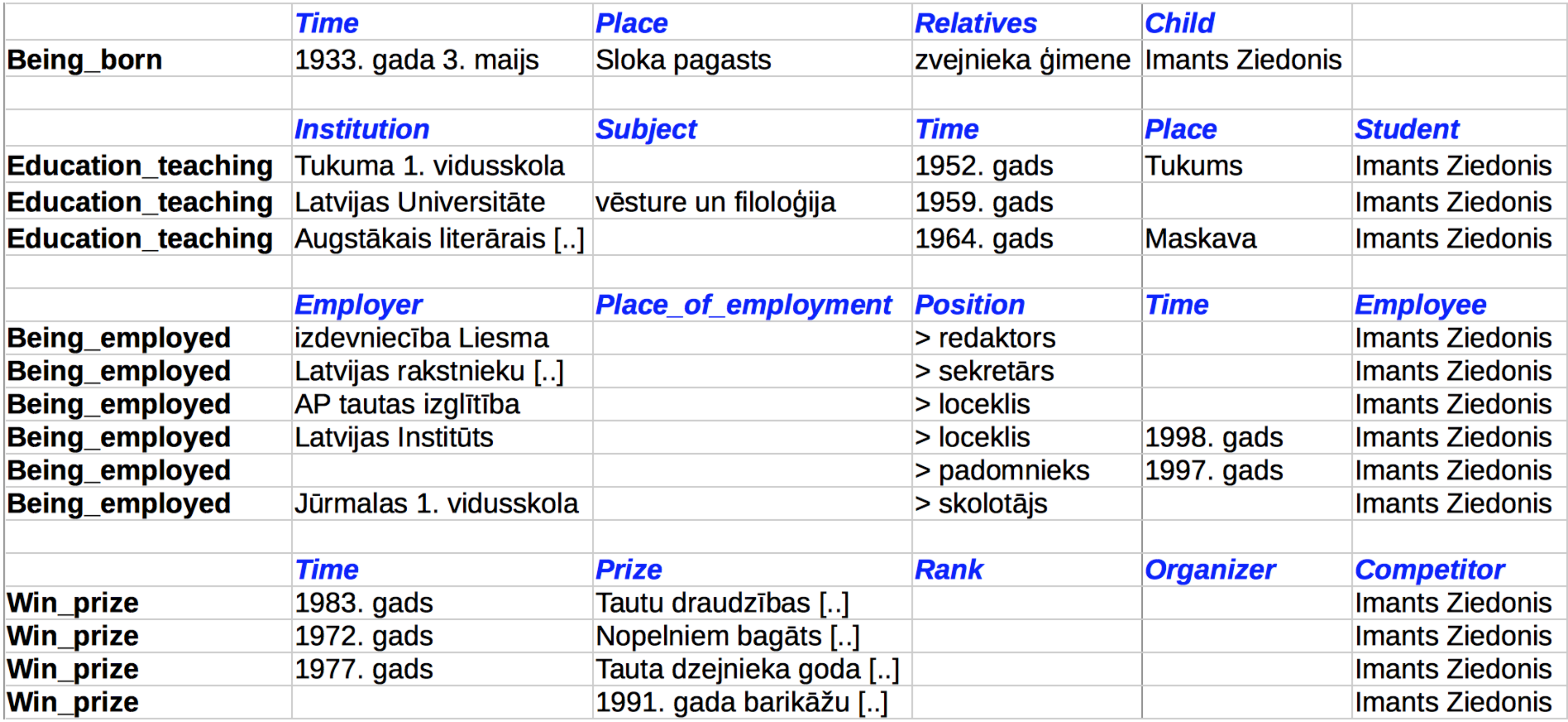
Semantic parsing (SRL)

- ToDo: functions that return frame elements,
  - a technical frame element for the target word,
  - decomposition of elements of type Adv
- Open issues:
  - A closed set of target verbs per frame
  - A closed set of prepositions per frame element (if realized as a PP)
  - Support for variable word order (Adv modifiers)
- Meanwhile, a statistical FrameNet parser can be used, e.g. for IE



http://demo.ark.cs.cmu.edu/parse

Natural language generation

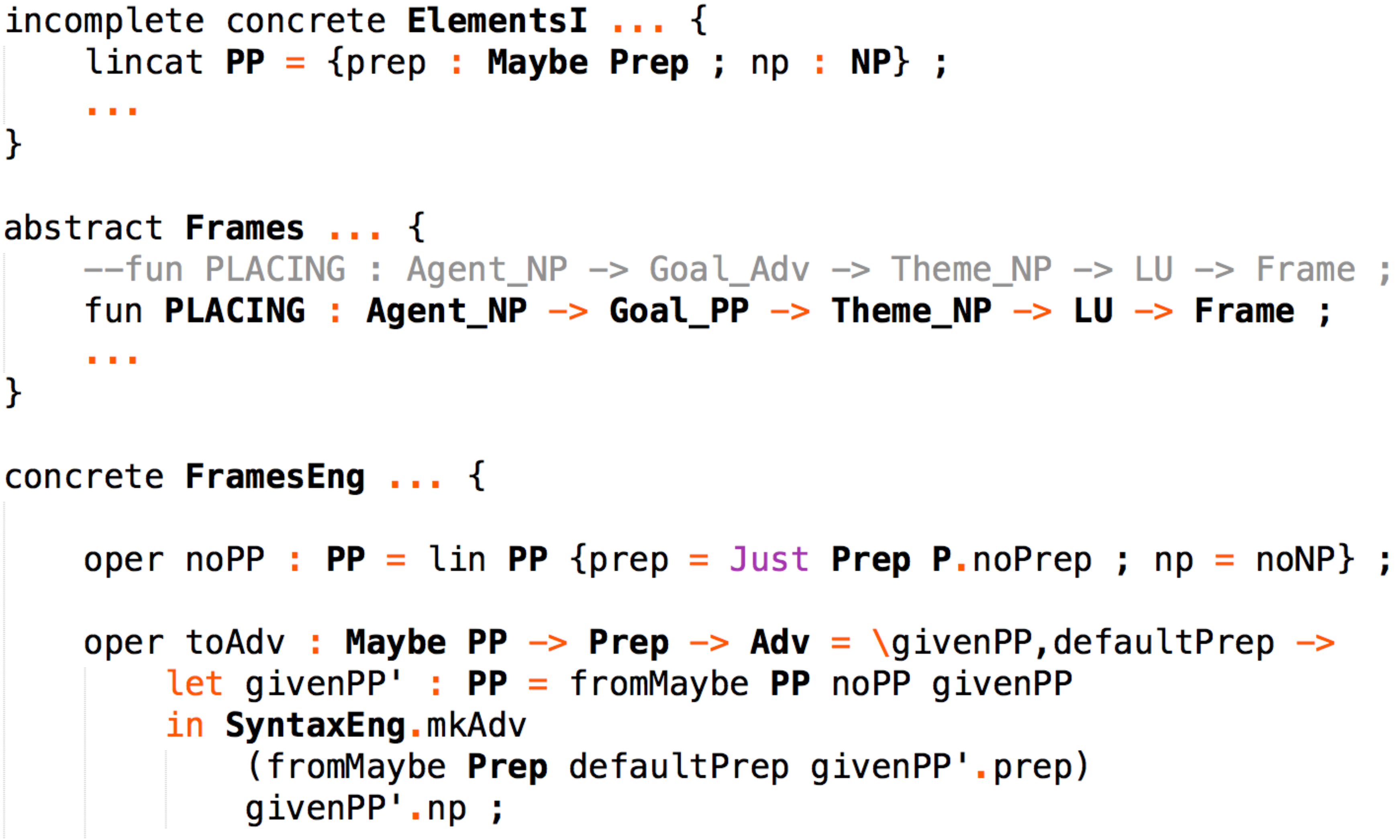


	Time	*Place*	*Relatives*	*Child*	
Being_born	1933. gada 3. maijs	Sloka pagasts	zvejnieka ģimene	Imants Ziedonis	
	Institution	*Subject*	*Time*	*Place*	*Student*
Education_teaching	Tukuma 1. vidusskola		1952. gads	Tukums	Imants Ziedonis
Education_teaching	Latvijas Universitāte	vēsture un filoloģija	1959. gads		Imants Ziedonis
Education_teaching	Augstākais literārais [..]		1964. gads	Maskava	Imants Ziedonis
	Employer	*Place_of_employment*	*Position*	*Time*	*Employee*
Being_employed	izdevniecība Liesma		> redaktors		Imants Ziedonis
Being_employed	Latvijas rakstnieku [..]		> sekretārs		Imants Ziedonis
Being_employed	AP tautas izglītība		> loceklis		Imants Ziedonis
Being_employed	Latvijas Institūts		> loceklis	1998. gads	Imants Ziedonis
Being_employed			> padomnieks	1997. gads	Imants Ziedonis
Being_employed	Jūrmalas 1. vidusskola		> skolotājs		Imants Ziedonis
	Time	*Prize*	*Rank*	*Organizer*	*Competitor*
Win_prize	1983. gads	Tautu draudzības [..]			Imants Ziedonis
Win_prize	1972. gads	Nopelniem bagāts [..]			Imants Ziedonis
Win_prize	1977. gads	Tauta dzejnieka goda [..]			Imants Ziedonis
Win_prize		1991. gada barikāžu [..]			Imants Ziedonis

E.g. given a DB of CV-style facts extracted from Lav newswire texts (using a statistical parser)
→ provide a multilingual NL interface

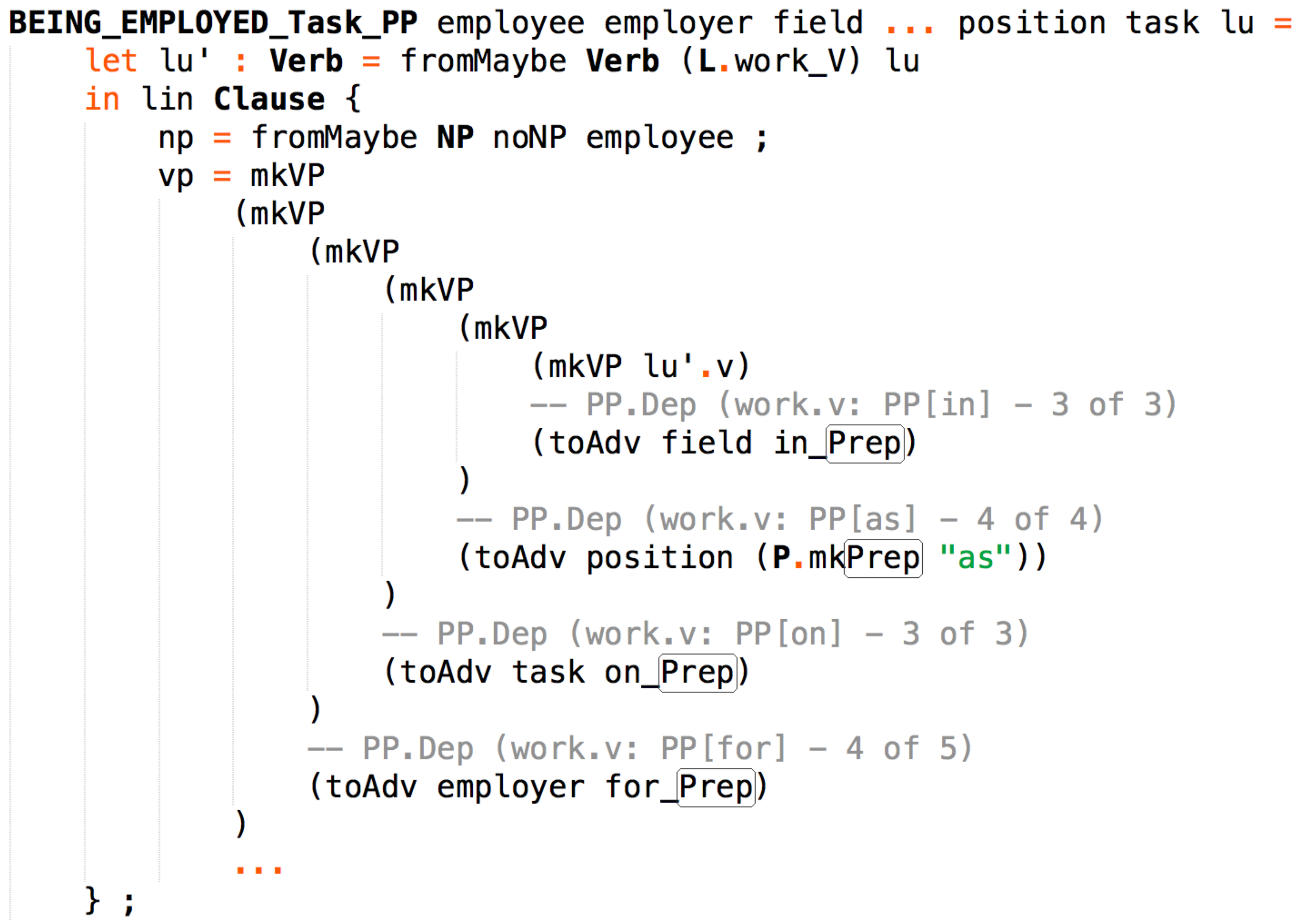
* Frames could have been triggered by nouns → paraphrasing using verbal constructions
* The original prepositions/cases might not be available → arguments vs. adjuncts
* Sentence planning and splitting, anaphora generation, parameter to change the voice etc.

Prototype #2: decomposing Adv



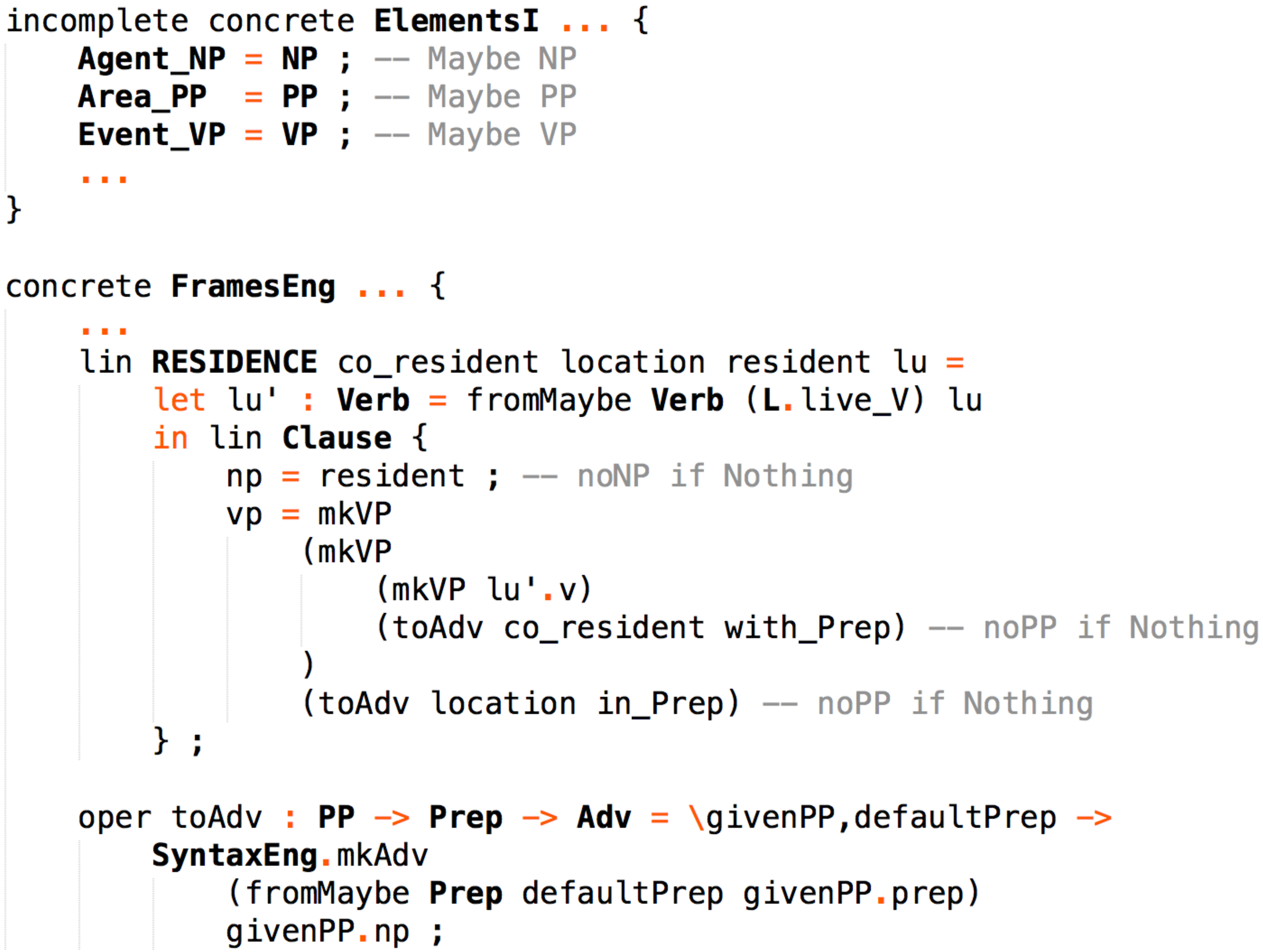
```
incomplete concrete ElementsI ... {
    lincat PP = {prep : Maybe Prep ; np : NP} ;
    ...
}

abstract Frames ... {
    --fun PLACING : Agent_NP -> Goal_Adv -> Theme_NP -> LU -> Frame ;
    fun PLACING : Agent_NP -> Goal_PP -> Theme_NP -> LU -> Frame ;
    ...
}

concrete FramesEng ... {

    oper noPP : PP = lin PP {prep = Just Prep P.noPrep ; np = noNP} ;

    oper toAdv : Maybe PP -> Prep -> Adv = \givenPP,defaultPrep ->
        let givenPP' : PP = fromMaybe PP noPP givenPP
        in SyntaxEng.mkAdv
            (fromMaybe Prep defaultPrep givenPP'.prep)
            givenPP'.np ;
```

Prototype #2: decomposing Adv



```
BEING_EMPLOYED_Task_PP employee employer field ... position task lu =
    let lu' : Verb = fromMaybe Verb (L.work_V) lu
    in lin Clause {
        np = fromMaybe NP noNP employee ;
        vp = mkVP
            (mkVP
                (mkVP
                    (mkVP
                        (mkVP
                            (mkVP lu'.v)
                            -- PP.Dep (work.v: PP[in] - 3 of 3)
                            (toAdv field in_Prep)
                        )
                        -- PP.Dep (work.v: PP[as] - 4 of 4)
                        (toAdv position (P.mkPrep "as"))
                    )
                    -- PP.Dep (work.v: PP[on] - 3 of 3)
                    (toAdv task on_Prep)
                )
                -- PP.Dep (work.v: PP[for] - 4 of 5)
                (toAdv employer for_Prep)
            )
            ...
    } ;
```

Prototype #2.1: minimizing Maybe



```
incomplete concrete ElementsI ... {
    Agent_NP = NP ; -- Maybe NP
    Area_PP  = PP ; -- Maybe PP
    Event_VP = VP ; -- Maybe VP
    ...
}

concrete FramesEng ... {
    ...
    lin RESIDENCE co_resident location resident lu =
        let lu' : Verb = fromMaybe Verb (L.live_V) lu
        in lin Clause {
            np = resident ; -- noNP if Nothing
            vp = mkVP
                (mkVP
                    (mkVP lu'.v)
                    (toAdv co_resident with_Prep) -- noPP if Nothing
                )
                (toAdv location in_Prep) -- noPP if Nothing
        } ;

    oper toAdv : PP -> Prep -> Adv = \givenPP,defaultPrep ->
        SyntaxEng.mkAdv
            (fromMaybe Prep defaultPrep givenPP.prep)
            givenPP.np ;
```

Conclusions and future directions

- FrameNet API would facilitate the development of certain GF application grammars
  - Frames can be specified in the functor of an application grammar
  - Resulting grammars would be more generic and easier to extend
- Language-specific FrameNet resource grammars can be acquired semi-automatically from FrameNet data that include mapping to syntactic patterns and statistics from FrameNet-annotated corpora
  - Frames might be implemented even in the functor of the FN library
- Language generation and semantic parsing directly with the FrameNet library (as a general application grammar)