Horn Binary Serialization Analysis

HCVS 2016

3rd Workshop
on Horn Clauses
for Verification and Synthesis

Gabriele Paganelli
https://gapag.noblogs.org/
gapag@distruzione.org

Length	Type	***Data***	CRC

0 3 4 7 8 N N+1 N+4

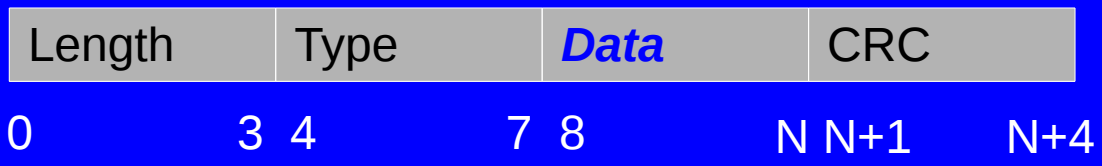
Type	***Data***	Length	CRC

0 3 4 N N+1 N+4 N+5 N+8

Cat.png

Contribution

- **Given** a layout specification,



Is there a parser

that can parse any instance stream?

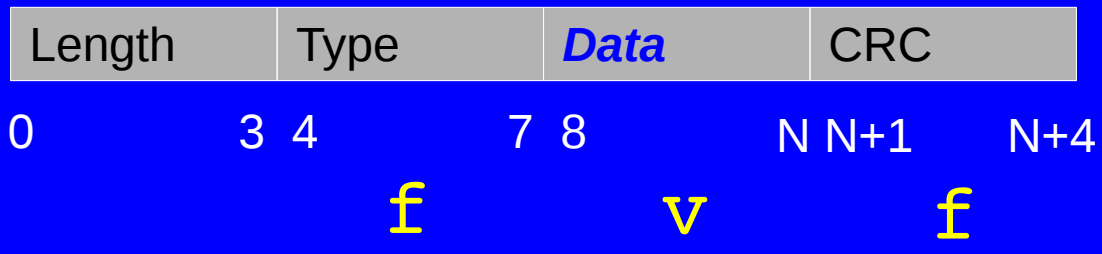
Or: *is the layout deserializable?*

***In practice:* describe a left-to-right parser behaviour using Horn clauses and forward chaining**

1:Formalize a layout specification

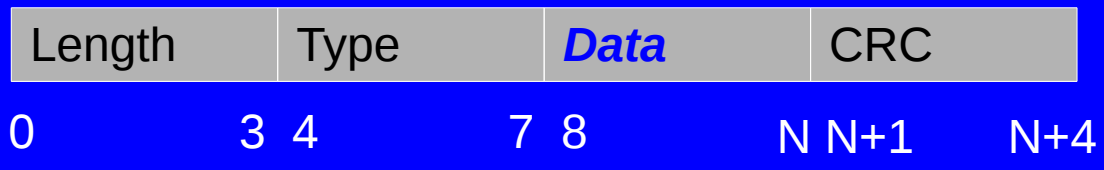
- `f [Id]` : fixed length field
- `v [Id]` : variable length field (*varfield*)
- `(Id' → ß)[Id]`: pointer field
  - `Id'`: offset
  - `ß`: span

`(Length→4)Length`



2:Give a name to all fields

(Length → 4)Length_0 f_1 v_2 f_3

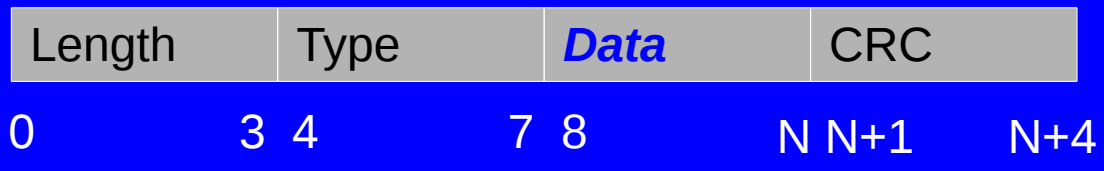


3:Formalize parser's knowledge

- ***The parser knows...***
- *Beg(i)* : where field *i* begins
- *Len(i)* : field *i*'s length
- *Ptr(o,s,i)*: field *i* is a pointer,
  with offset at *o*
  and spanning *s* fields
- *Val(i)*: field *i*'s contents.

4a:Formalize parser's behaviour

$(\mathtt{Length} \rightarrow 4)\mathtt{Length}_0\ \mathtt{f}_1\ \mathtt{v}_2\ \mathtt{f}_3$

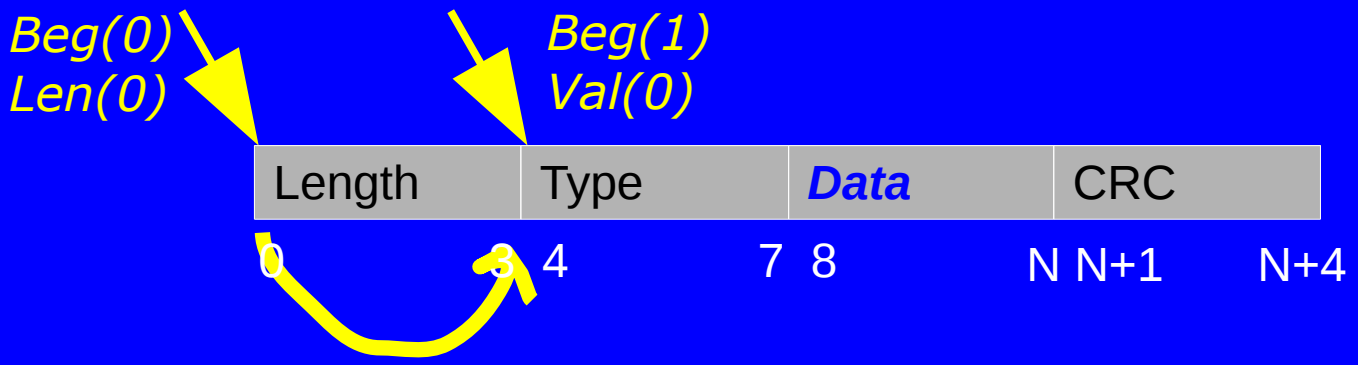


True ⇒ Beg(0)
True ⇒ Len(0)
True ⇒ Len(1)
True ⇒ Len(3)
True ⇒ Ptr(0,4,0)

4b:Formalize parser's behaviour

- $Beg(i) \wedge Len(i) \Rightarrow Beg(i+1) \wedge Val(i)$ forward
- $Beg(i+1) \wedge Len(i) \Rightarrow Beg(i) \wedge Val(i)$ backward

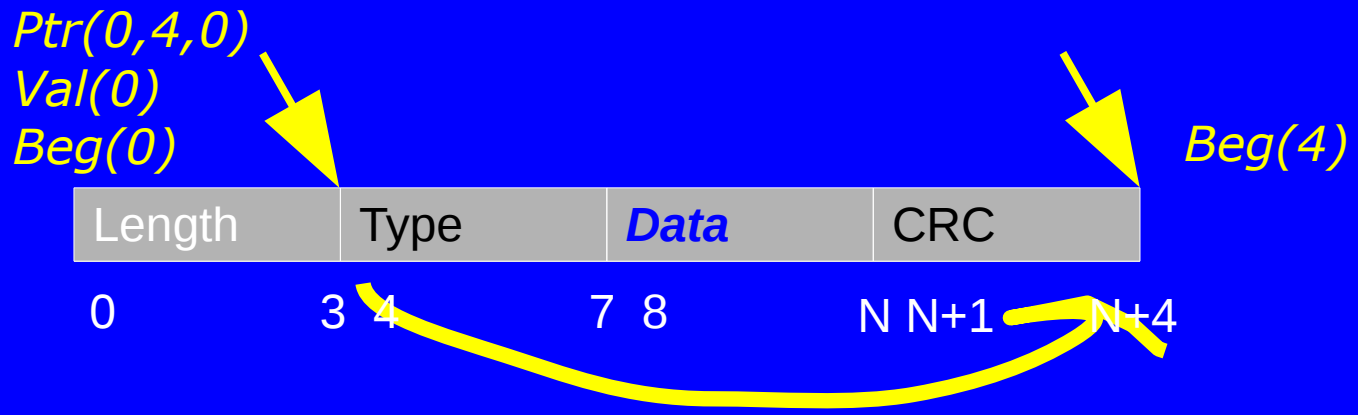
- Read a field backward or forward.



4c:Formalize parser's behaviour

- $Ptr(o,s,i) \wedge Val(i) \wedge Beg(o) \Rightarrow Beg(o+s)$ Jump right
- $Ptr(o,s,i) \wedge Val(i) \wedge Beg(o+s) \Rightarrow Beg(o)$ Jump left

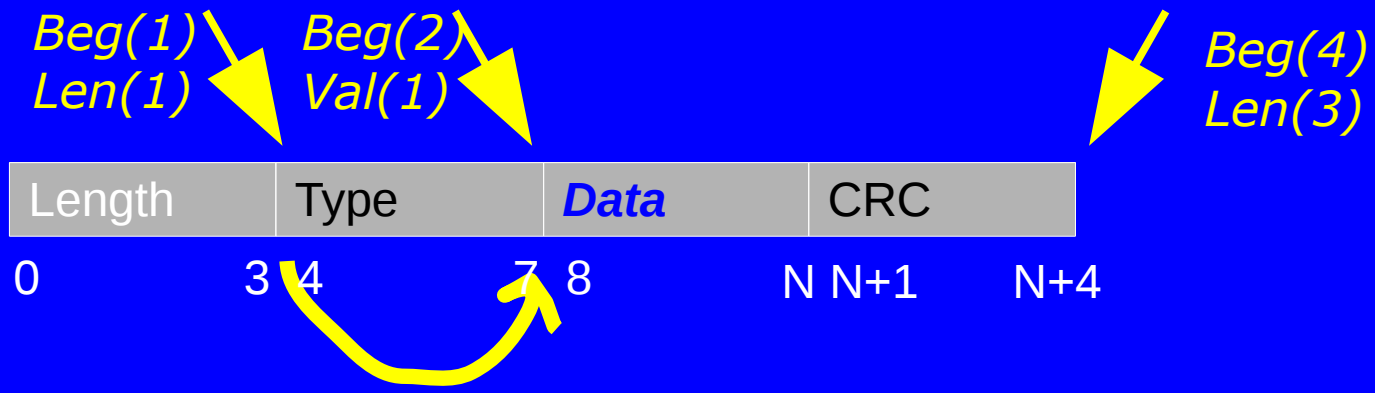
- Follow a pointer backward or forward.



4cc:(example, continued)

- $Beg(i) \wedge Len(i) \Rightarrow Beg(i+1) \wedge Val(i)$ forward
- $Beg(i+1) \wedge Len(i) \Rightarrow Beg(i) \wedge Val(i)$ backward

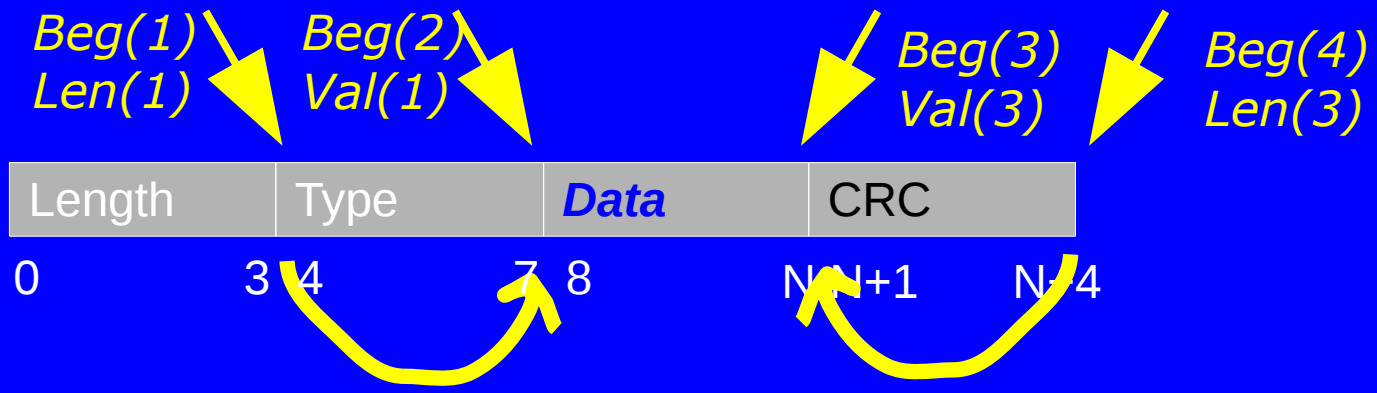
- Read a field backward or forward.



4cc:(example, continued)

- $Beg(i) \wedge Len(i) \Rightarrow Beg(i+1) \wedge Val(i)$ forward
- $Beg(i+1) \wedge Len(i) \Rightarrow Beg(i) \wedge Val(i)$ backward

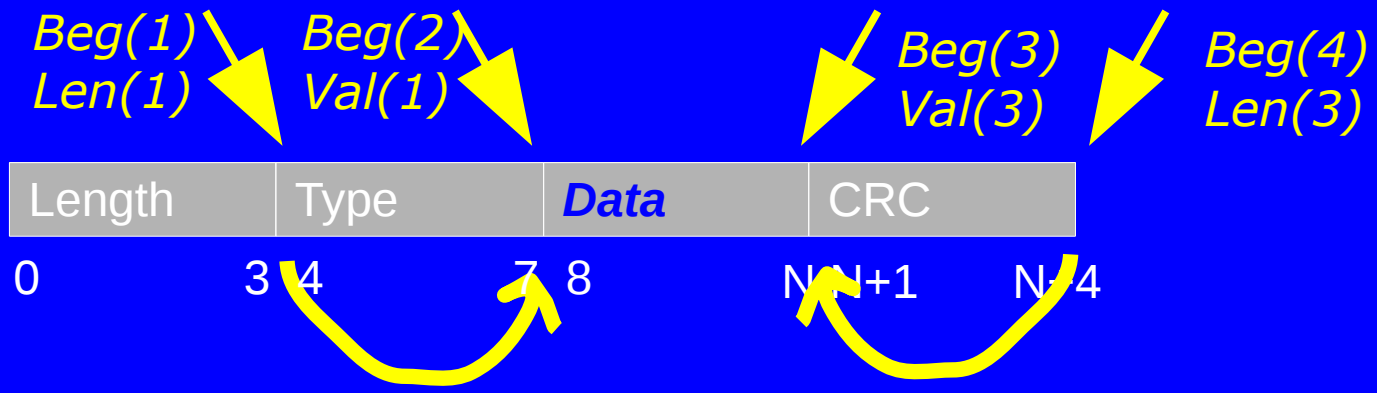
- Read a field backward or forward.



4cc:(example, continued)

- $Beg(i) \wedge Len(i) \Rightarrow Beg(i+1) \wedge Val(i)$ forward
- $Beg(i+1) \wedge Len(i) \Rightarrow Beg(i) \wedge Val(i)$ backward

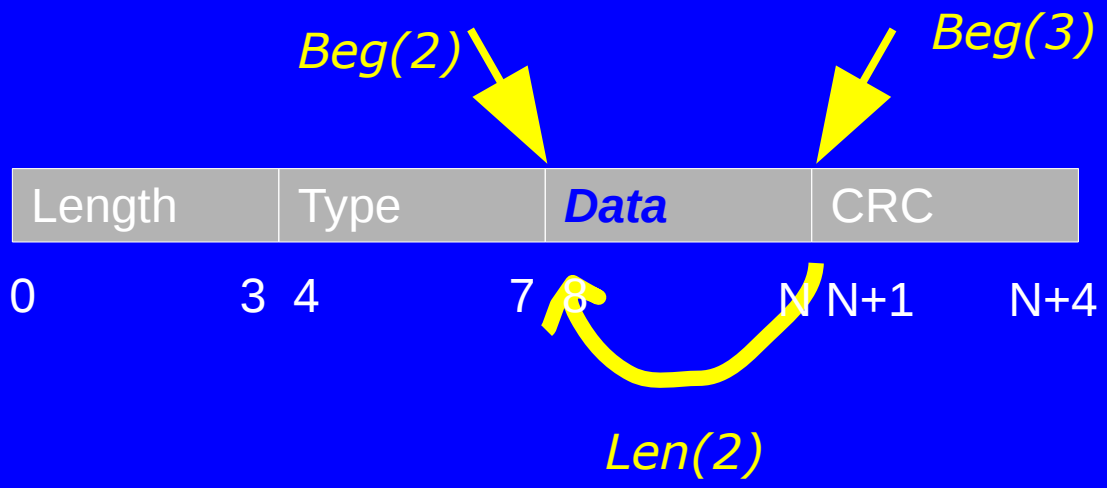
- Read a field backward or forward.



4d:Formalize parser's behaviour

- $Beg(i) \wedge Beg(i+1) \Rightarrow Len(i)$ join

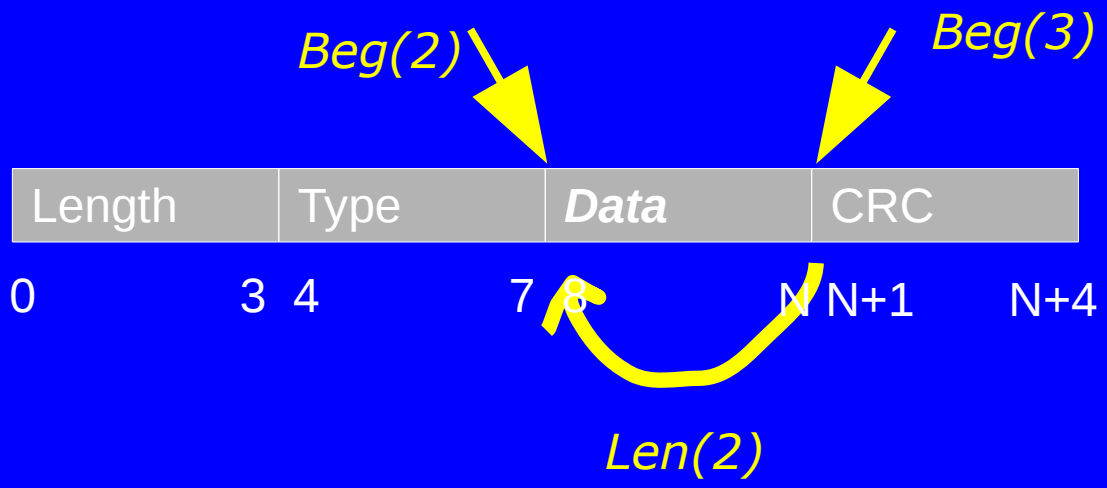
- Compute the length of a field.



4d:Formalize parser's behaviour

- $Beg(i) \wedge Beg(i+1) \Rightarrow Len(i)$ join

- Compute the length of a field.



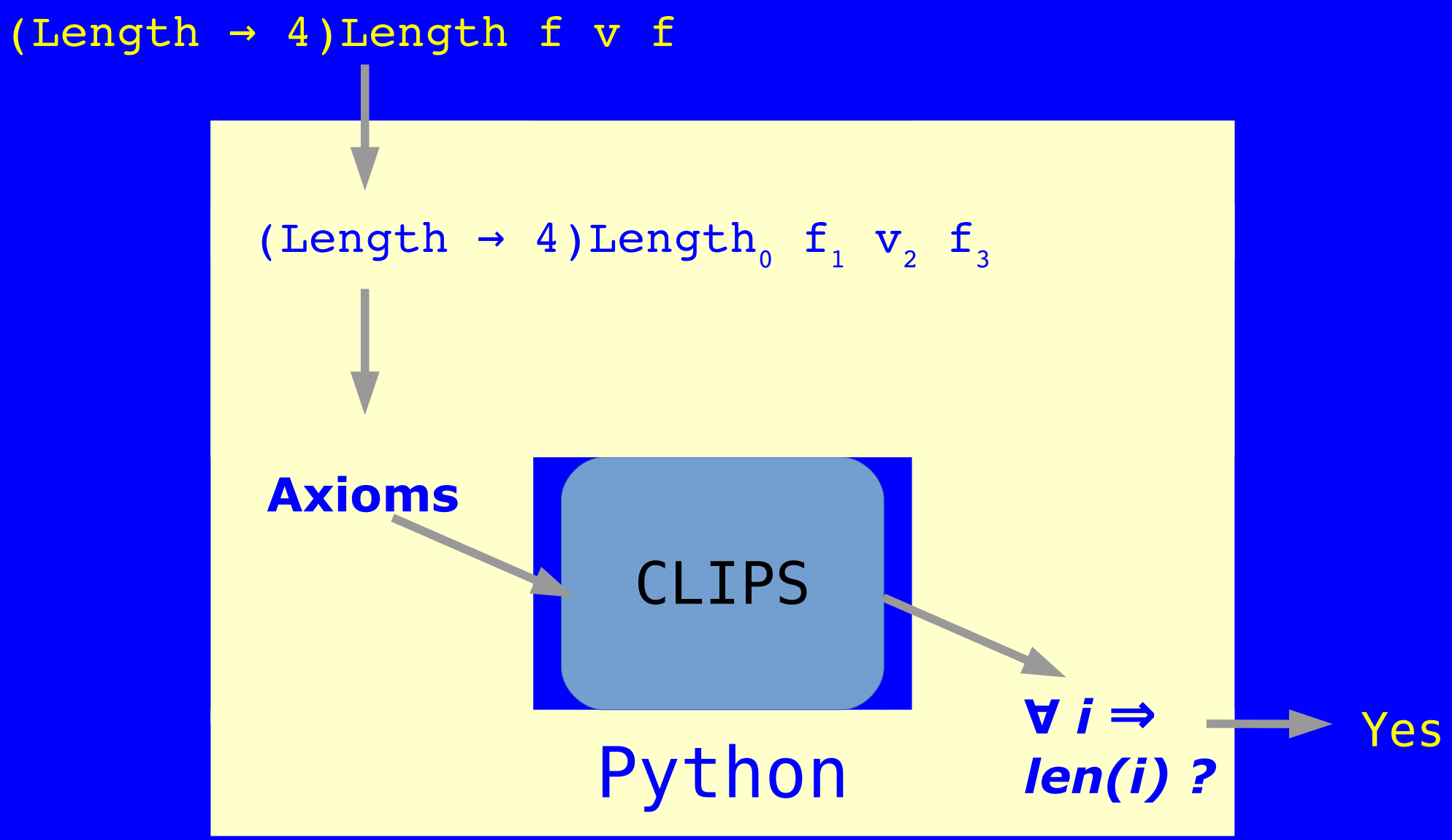
Deserializability Check Algorithm

- Transform a layout into a Horn KB: *O(3n)*
- Apply forward chaining: *O(3n)*
- Is *Len(i)* for all *i* in a layout in KB? *O(n)*

Yes: Layout is deserializable

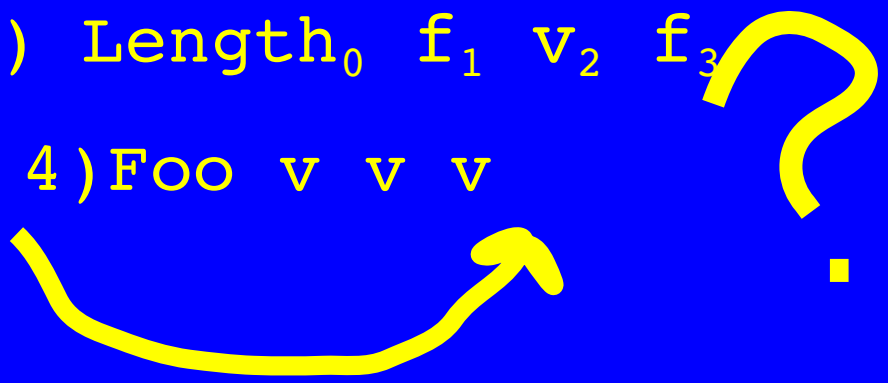
No: Layout is not deserializable.

Implementation



Necessary condition for deserialization

- If layout L is deserializable, THEN in L
  - for every v_i
  - There is a $(foo \rightarrow s)_p$, $xfoo_q$
    - Such that $q \leq i < q+s$
  - e.g. : `f v f`
  - `(Length → 4) Length`$_0$ `f`$_1$ `v`$_2$ `f`$_3$
  - But: `(Foo → 4)Foo v v v`



Repetition (Kleene star) : [ ]*

```
(Foo → 2)Foo f [f v f ]*
```

Repetition (Kleene star) : [ ]*

(Foo → 2)Foo f [f v f]*

$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ ]^*_2$

List labels instead of natural labels

Non valid layout specs

- $(\texttt{Bar} \rightarrow 2)_0[\texttt{fBar v f}]^*_1$
  - Referencing into an inner scope.

Pointers
cannot
offset into inner scopes

$(\texttt{Bar} \rightarrow 2)_0[$
fBar v f fBar v f fBar v f fBar v f
fBar v f fBar v f fBar v f fBar v f fBar v f
fBar v f fBar v f fBar v f fBar v f fBar v f
fBar v f fBar v f fBar v f fBar v f fBar v f
fBar v f fBar v f fBar v f fBar v f fBar v f
fBar v f fBar v f fBar v f fBar v f fBar v f

[ ]*: Predicates

(Foo → 2)Foo_0 f_1 [$f_{2.0}$ $v_{2.1}$ $f_{2.2}$]*$_2$

- *Rep(i,l)* : field *i* is a repetition containing *l* fields
- *Replen(i)* : the parser knows repetition *i*'s length

What about the axioms?

- Lifted to the list label level:
- e.g:

$Beg(b.a) \wedge Beg(b.a+1) \Rightarrow Len(b.a)$ join

$Ptr(b.a,s,i) \wedge Val(i) \wedge Beg(b.a) \Rightarrow Beg(b.a+s)$ Jump right

b,i :list

*s,a :*natural

[ ]*: Axioms

$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ \mathrm{f}_1\ [\mathrm{f}_{2.0}\ \mathrm{v}_{2.1}\ \mathrm{f}_{2.2}\ ]^*_2$

- *True* ⇒ *Rep(i,l)*

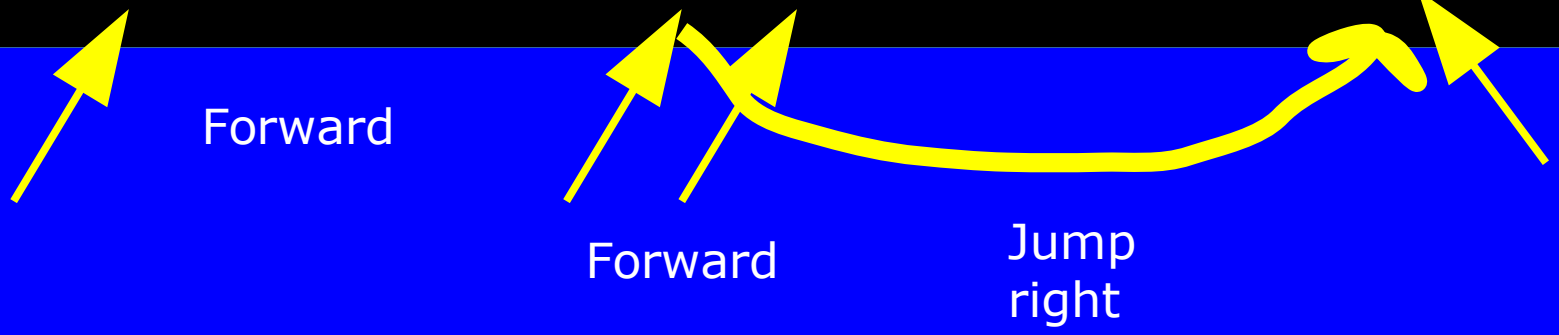
  in this layout: *True* ⇒ *Rep(2,3)*

- *Rep(b.a,l)* ∧ *Beg(b.a)* ∧ *Beg(b.a+1)* ⇒ *RepLen(b.a)*

- *Rep(b.a,l)* ∧ *Beg(b.a)* ⇒ *Beg(b.a.0)*
- *Rep(b.a,l)* ∧ *Beg(b.a+1)* ⇒ *Beg(b.a.l)*

PITFALL!

$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ ]^*_2$



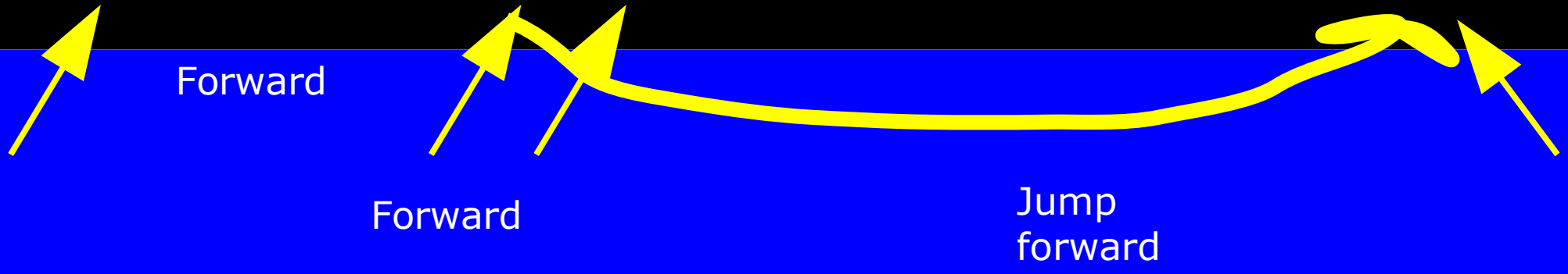
PITFALL!

$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ ]^*_2$

- *Rep(2,3) ∧ Beg(2) ⇒ Beg(2.0)*
- *Rep(2,3) ∧ Beg(3) ⇒ Beg(2.3)*

PITFALL!

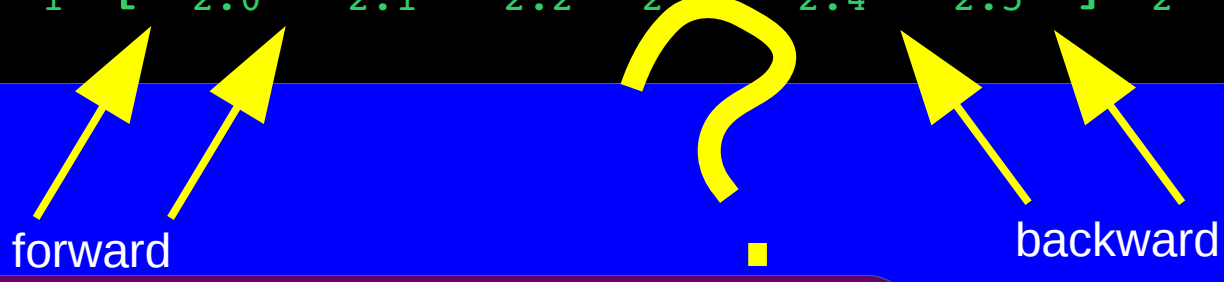
$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ f_{2.3}\ v_{2.4}\ f_{2.5}\ ]^*_2$



$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ ]^*_2$

PITFALL!

$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ f_{2.3}\ v_{2.4}\ f_{2.5}\ ]^*_2$



- *Rep(2,3) ∧ Beg(2) ⇒ Beg(2.0)*
- *Rep(2,3) ∧ Beg(3) ⇒ Beg(2.3)*

$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ ]^*_2$

Dirty trick

$$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ ]^*_2$$

$$(\mathrm{Foo} \rightarrow 2)\mathrm{Foo}_0\ f_1\ [f_{2.0}\ v_{2.1}\ f_{2.2}\ f_{2.3}\ v_{2.4}\ f_{2.5}]^*_2$$

Take each repetition field and double its content.

If L2 is (not) deserializable
Then all LN>2 are (not) deserializable

rormally guaranteed:
If L2 is deserializable,
Then L is deserializable

L $(A \to 2)A_0\ f_1\ [f_{2.0}$

$[(B \to 1)B,\ v]^*_{2.1}$

$f_{2.2}]^*_2$

$(A \to 2)A_0\ f_1\ [f_{2.0}$ L2

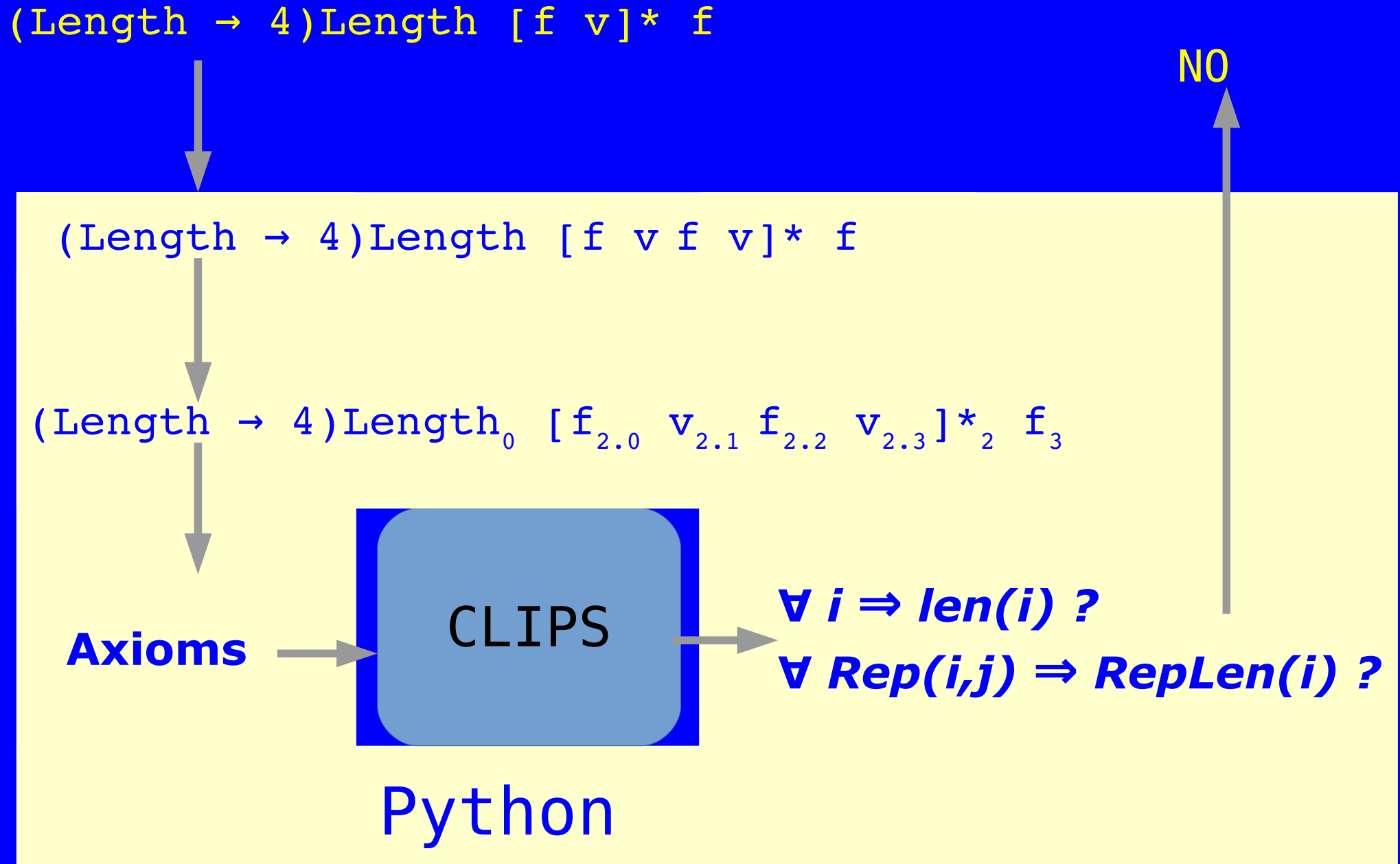
$[(B \to 1)B\ v\ (C \to 1)C\ v]^*_{2.1}$

$f_{2.2}\ f_{2.3}\ [D \to 1)D\ v]^*_{2.4}\ f_{2.5}]^*_2$

O(kn)

Axioms are left undisturbed.

Implementation



Intended application areas

- Serialization libraries
- Data definition language C!C
  - Rule-based parser generation?
  - Associate to each proof of deserializability a parser.

Related work

- Erlang, haskell, c...
- Pads
- Protocol buffers, avro, cap'n'proto, bson...

Summary

- Axiomatization of left-to-right stream parsing
- Implementation: Python+CLIPS
- Interesting results:
  - Necessary condition for deserializability
  - Doubling repetitions

Gabriele Paganelli
https://github.com/gapag/horn-binary-deserialization
https://gapag.noblogs.org/
gapag@distruzione.org