KU LEUVEN

DTAI
Declarative Languages & Artificial Intelligence

LEUVEN.AI

Flanders AI
Research Program

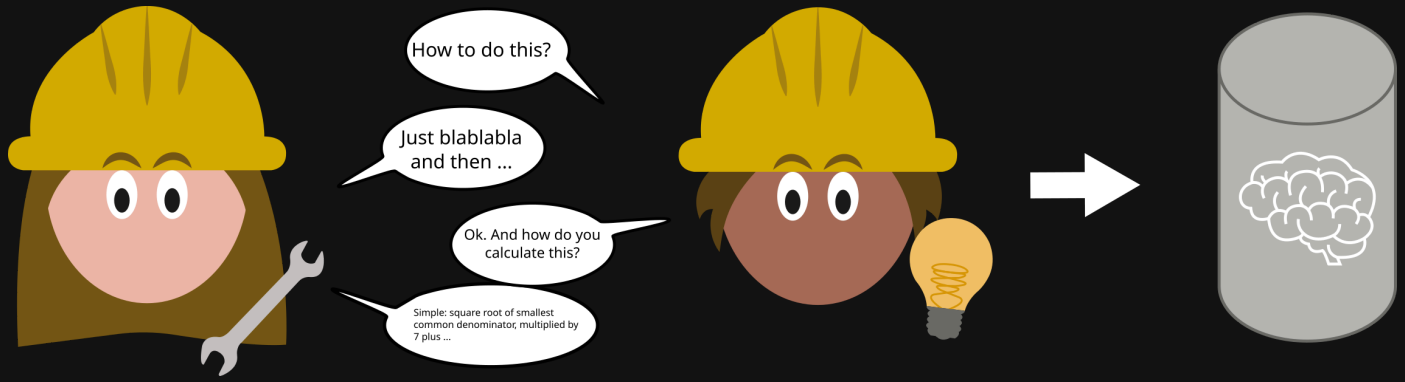
cDMN for User-Friendly ASP

Simon Vandevelde, TAASP 2026

Knowledge representation is hard

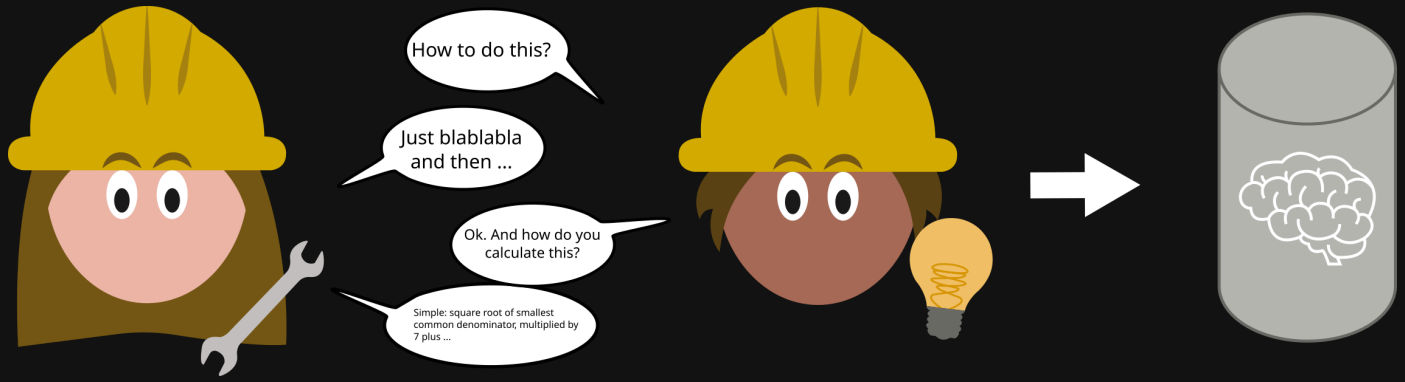
Knowledge representation is hard

Knowledge Acquisition Bottleneck



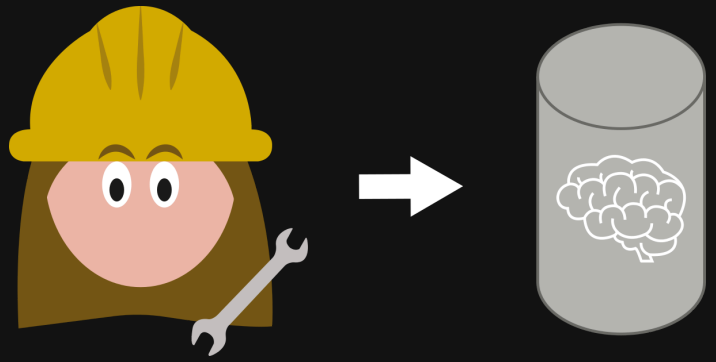
Knowledge representation is hard

Knowledge Acquisition Bottleneck

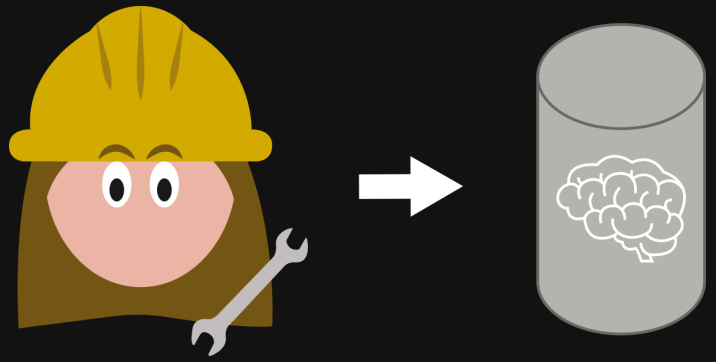


🏆 let **non-experts** formalise their own knowledge,
without a knowledge engineer. 🏆

🏆 Easier said than done?



🏆 Easier said than done?



Requires a language that is

1. Expressive enough
2. Sufficiently user-friendly
3. Formal

Decision Model and Notation

OMG standard

Decision Model and Notation

OMG standard

- Decision tables

Decide method to use			
U	weight	hole	method
1	≤ 10	Yes	Nails
2	(10..25]	Yes	Screws
3	≤ 15	No	Glue

Decision Model and Notation

OMG standard

- Decision tables
- 🎯 readable **and** writeable for *everyone* in the decision modelling process

Decide method to use			
U	weight	hole	method
1	≤ 10	Yes	Nails
2	(10..25]	Yes	Screws
3	≤ 15	No	Glue

Decision Model and Notation

OMG standard

- Decision tables
- 🎯 readable **and** writeable for *everyone* in the decision modelling process
- Meaning given by **Hit policy**

Decide method to use			
U	weight	hole	method
1	≤ 10	Yes	Nails
2	(10..25]	Yes	Screws
3	≤ 15	No	Glue

Decision Model and Notation

OMG standard

- Decision tables
- 🎯 readable **and** writeable for *everyone* in the decision modelling process
- Meaning given by **Hit policy**
- Can be chained together

Decide if hole is possible		
U	wall	hole
1	Wood, Brick	Yes
2	Tile	No

Decide method to use			
U	weight	hole	method
1	≤ 10	Yes	Nails
2	(10..25]	Yes	Screws
3	≤ 15	No	Glue

Sufficiently expressive?

→ Every decision table must **precisely define** a symbol

Sufficiently expressive?

→ Every decision table must **precisely define** a symbol

✔ "if the weight is 15, you must use glue"

Sufficiently expressive?

→ Every decision table must **precisely define** a symbol

✔️ "if the weight is 15, you must use glue"

❌ "if we use glue, the weight must be below 15"

constraint Decision Model and Notation

constraint Decision Model and Notation

Max load		
E*	method	weight
1	Nails	≤ 10
2	Screws	≤ 25
3	Glue	≤ 15

- **Constraint tables**: hit policy `E*`

constraint Decision Model and Notation

Max load		
E*	method	weight
1	Nails	≤ 10
2	Screws	≤ 25
3	Glue	≤ 15

- **Constraint tables**: hit policy `E*`
- Additionally:
  - higher-ary symbols
  - quantification
  - and more

cDMN: map coloring

cDMN: map coloring

Type		
Name	DataType	PossibleValues
Country	string	BE, FR, LU, NL, DE, DK, UK
Color	string	Red, Blue, Green, Purple

cDMN: map coloring

Type		
Name	DataType	PossibleValues
Country	string	BE, FR, LU, NL, DE, DK, UK
Color	string	Red, Blue, Green, Purple

Function	
Name	DataType
color of Country	Color

cDMN: map coloring

Type		
Name	DataType	PossibleValues
Country	string	BE, FR, LU, NL, DE, DK, UK
Color	string	Red, Blue, Green, Purple

Function	
Name	DataType
color of Country	Color

Relation
Name
Country borders Country

cDMN: map coloring

Type		
Name	DataType	PossibleValues
Country	string	BE, FR, LU, NL, DE, DK, UK
Color	string	Red, Blue, Green, Purple

Function	
Name	DataType
color of Country	Color

Relation
Name
Country borders Country

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	-	Yes	not(color of c2)

cDMN: map coloring

Type		
Name	DataType	PossibleValues
Country	string	BE, FR, LU, NL, DE, DK, UK
Color	string	Red, Blue, Green, Purple

Function	
Name	DataType
color of Country	Color

Relation
Name
Country borders Country

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	-	Yes	not(color of c2)

$$\forall c_1, c_2 \in Country : borders(c1, c2) \Rightarrow color_of(c_1) \neq color_of(c_2)$$

cDMN semantics, briefly

- A decision table is a tuple $DT = \langle I, O, R, H \rangle$
- row cells contain **S-FEEL** expressions
- each cell corresponds to a formula $F_{k,j}(x)$ with one free variable

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

cDMN semantics, briefly

- A decision table is a tuple $DT = \langle I, O, R, H \rangle$
- row cells contain **S-FEEL** expressions
- each cell corresponds to a formula $F_{k,j}(x)$ with one free variable

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

- Variable-introducing columns: $\forall c_1 \in Country, c_2 \in Country$

cDMN semantics, briefly

- A decision table is a tuple $DT = \langle I, O, R, H \rangle$
- row cells contain **S-FEEL** expressions
- each cell corresponds to a formula $F_{k,j}(x)$ with one free variable

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

- Variable-introducing columns: $\forall c_1 \in Country, c_2 \in Country$
- True

cDMN semantics, briefly

- A decision table is a tuple $DT = \langle I, O, R, H \rangle$
- row cells contain **S-FEEL** expressions
- each cell corresponds to a formula $F_{k,j}(x)$ with one free variable

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

- Variable-introducing columns: $\forall c_1 \in Country, c_2 \in Country$
- True
- $c_2 \neq c_1$

cDMN semantics, briefly

- A decision table is a tuple $DT = \langle I, O, R, H \rangle$
- row cells contain **S-FEEL** expressions
- each cell corresponds to a formula $F_{k,j}(x)$ with one free variable

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

- Variable-introducing columns: $\forall c_1 \in Country, c_2 \in Country$
- True
- $c_2 \neq c_1$
- $borders(c_1, c_2)$

cDMN semantics, briefly

- A decision table is a tuple $DT = \langle I, O, R, H \rangle$
- row cells contain **S-FEEL** expressions
- each cell corresponds to a formula $F_{k,j}(x)$ with one free variable

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

- Variable-introducing columns: $\forall c_1 \in Country, c_2 \in Country$
- True
- $c_2 \neq c_1$
- $borders(c_1, c_2)$
- $color_of(c_1) \neq color_of(c_2)$

cDMN semantics, briefly (contd.)

cDMN semantics, briefly (contd.)

Decision tables (`U`) → *definitional rules* in **FO(ID)**, together forming a complete definition:

$$\forall \nu(V) : t_\nu\big(F_{|I|+1,j}(t_\nu(O))\big) \leftarrow \bigwedge_{k=1}^{|I|} t_\nu\big(F_{j,k}(t_\nu(I_k))\big)$$

cDMN semantics, briefly (contd.)

Decision tables (`U`) → *definitional rules* in **FO(ID)**, together forming a complete definition:

$$\forall \nu(V) : t_\nu\big(F_{|I|+1,j}(t_\nu(O))\big) \leftarrow \bigwedge_{k=1}^{|I|} t_\nu\big(F_{j,k}(t_\nu(I_k))\big)$$

Constraint tables (`E*`) → *material implications*:

$$\forall \nu(V) : \Big(\bigwedge_{k=1}^{|I|} t_\nu\big(F_{k,j}(t_\nu(I_k))\big) \Rightarrow t_\nu\big(F_{|I|+1,j}(t_\nu(O))\big)\Big)$$

U vs E*

U vs E*

$$
\begin{aligned}
&\{\\
&method() = Nails \leftarrow weight() \leq 10 \wedge hole().\\
&method() = Screws \leftarrow 10 < weight() \leq 25 \wedge hole().\\
&method() = Glue \leftarrow weight() \leq 15 \wedge \neg hole().\\
&\}
\end{aligned}
$$

Decide method to use			
U	weight	hole	method
1	≤ 10	Yes	Nails
2	(10..25]	Yes	Screws
3	≤ 15	No	Glue

U vs E*

$$
\begin{aligned}
&\{\\
&method() = Nails \leftarrow weight() \leq 10 \wedge hole().\\
&method() = Screws \leftarrow 10 < weight() \leq 25 \wedge hole().\\
&method() = Glue \leftarrow weight() \leq 15 \wedge \neg hole().\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&(weight \leq 10 \wedge hole() \Rightarrow method() = Nails)\\
&\wedge(weight \in (10..25] \wedge hole() \Rightarrow method() = Screws)\\
&\wedge(weight() \leq 15 \wedge \neg hole() \Rightarrow method() = Glue)
\end{aligned}
$$

Decide method to use			
U	weight	hole	method
1	≤ 10	Yes	Nails
2	(10..25]	Yes	Screws
3	≤ 15	No	Glue

Decide method to use			
E*	weight	hole	method
1	≤ 10	Yes	Nails
2	(10..25]	Yes	Screws
3	≤ 15	No	Glue

Running example: modified map coloring

Function	
Name	DataType
color of Country	Color
nb nbrs of Country	Int

Relation
Name
Country borders Country
Country is isolated

Define island			
U	Country	nb nbrs of Country	Country is isolated
1	-	0	Yes
2	-	1	Yes

Count neighbors				
C+	Country called c1	Country called c2	c1 borders c2	nb nbrs of c1
1	-	-	Yes	1

1. Glossary

Each **type** becomes a unary predicate

```
bool(yes; no).
int(-100..100).
country(be; fr; lu; nl; de; dk; uk).
color(red; blue; green; purple).
```

1. Glossary

Each **type** becomes a unary predicate

```
bool(yes; no).
int(-100..100).
country(be; fr; lu; nl; de; dk; uk).
color(red; blue; green; purple).
```

Functions and predicates become *functional terms*, introduced via `symbol/1` and `domain/2`.

```
symbol(color_of_country(X0)) :- country(X0).
domain(color_of_country(X0), Y) :-
    country(X0), color(Y).

symbol(nb_nbrs_of_country(X0)) :- country(X0).
domain(nb_nbrs_of_country(X0), Y) :-
    country(X0), int(Y).

symbol(country_borders_country(X0, X1)) :-
    country(X0), country(X1).
domain(country_borders_country(X0, X1), Y) :-
    country(X0), country(X1), bool(Y).
```

1. Glossary

Each **type** becomes a unary predicate

```
bool(yes; no).
int(-100..100).
country(be; fr; lu; nl; de; dk; uk).
color(red; blue; green; purple).
```

Functions and predicates become *functional terms*, introduced via `symbol/1` and `domain/2`.

```
symbol(color_of_country(X0)) :- country(X0).
domain(color_of_country(X0), Y) :-
    country(X0), color(Y).

symbol(nb_nbrs_of_country(X0)) :- country(X0).
domain(nb_nbrs_of_country(X0), Y) :-
    country(X0), int(Y).

symbol(country_borders_country(X0, X1)) :-
    country(X0), country(X1).
domain(country_borders_country(X0, X1), Y) :-
    country(X0), country(X1), bool(Y).
```

`value/2` holds the value of each functional term

```
1 { value(Y, X): domain(Y, X) } 1 :- symbol(Y).
```

2. Constraint tables

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

2. Constraint tables

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

Each row becomes an **integrity constraint**

```
:- not V1 != V2, value(color_of_country(C1), V1),
                 value(color_of_country(C2), V2),
                 C2 != C1, V0 = yes,
                 value(country_borders_country(C1, C2), V0).
```

3. Decision tables

Define island			
U	Country	nb nbrs of Country	Country is isolated
1	-	0	Yes
2	-	1	Yes

3. Decision tables

Define island			
U	Country	nb nbrs of Country	Country is isolated
1	-	0	Yes
2	-	1	Yes

```
row(1, 0, C, yes) :- value(nb_nbrs_of_country(C), 0).
row(1, 1, C, yes) :- value(nb_nbrs_of_country(C), 1).
```

- `row(r, t, o1, ..., on, v)` tracks which row `r` of table `t` fires, for which quantification variables and output value

3. Decision tables

Define island			
U	Country	nb nbrs of Country	Country is isolated
1	-	0	Yes
2	-	1	Yes

```
row(1, 0, C, yes) :- value(nb_nbrs_of_country(C), 0).
row(1, 1, C, yes) :- value(nb_nbrs_of_country(C), 1).
```

```
:- not value(country_is_isolated(C), V), row(1, 0, C, V).
:- not value(country_is_isolated(C), V), row(1, 1, C, V).
```

- `row(r, t, o1, ..., on, v)` tracks which row `r` of table `t` fires, for which quantification variables and output value
- Decision tables over predicates only list the positive cases

3. Decision tables

Define island			
U	Country	nb nbrs of Country	Country is isolated
1	-	0	Yes
2	-	1	Yes

- `row(r, t, o1, ..., on, v)` tracks which row `r` of table `t` fires, for which quantification variables and output value
- Decision tables over predicates only list the positive cases

```
row(1, 0, C, yes) :- value(nb_nbrs_of_country(C), 0).
row(1, 1, C, yes) :- value(nb_nbrs_of_country(C), 1).
```

```
:- not value(country_is_isolated(C), V), row(1, 0, C, V).
:- not value(country_is_isolated(C), V), row(1, 1, C, V).
```

Plus the **default** `No`

```
:- not value(country_is_isolated(C), no),
   not row(1, 0, C, yes),
   not row(1, 1, C, yes),
   country(C).
```

4. Sum tables

Count neighbors				
C+	Country called c1	Country called c2	c1 borders c2	nb nbrs of c1
1	-	-	Yes	1

4. Sum tables

Count neighbors				
C+	Country called c1	Country called c2	c1 borders c2	nb nbrs of c1
1	-	-	Yes	1

- Variables in the **output header** determine the *grouping*: one sum per combination of their values (here: per `C1`)

```
V3 = S :-
    S = #sum{ 1, C2 :
        V0 = yes,
        value(country_borders_country(C1, C2), V0) },
    value(nb_nbrs_of_country(C1), V3).
```

4. Sum tables

Count neighbors				
C+	Country called c1	Country called c2	c1 borders c2	nb nbrs of c1
1	-	-	Yes	1

- Variables in the **output header** determine the *grouping*: one sum per combination of their values (here: per `C1`)
- All other variables are quantified inside the aggregate's body, together with the input columns

```
V3 = S :-
    S = #sum{ 1, C2 :
        V0 = yes,
        value(country_borders_country(C1, C2), V0) },
    value(nb_nbrs_of_country(C1), V3).
```

4. Sum tables

Count neighbors				
C+	Country called c1	Country called c2	c1 borders c2	nb nbrs of c1
1	-	-	Yes	1

- Variables in the **output header** determine the *grouping*: one sum per combination of their values (here: per `C1`)
- All other variables are quantified inside the aggregate's body, together with the input columns
- The **output cell** is the aggregate's term

```
V3 = S :-
    S = #sum{ 1, C2 :
        V0 = yes,
        value(country_borders_country(C1, C2), V0) },
    value(nb_nbrs_of_country(C1), V3).
```

4. Sum tables

Count neighbors				
C+	Country called c1	Country called c2	c1 borders c2	nb nbrs of c1
1	-	-	Yes	1

- Variables in the **output header** determine the *grouping*: one sum per combination of their values (here: per `C1`)
- All other variables are quantified inside the aggregate's body, together with the input columns
- The **output cell** is the aggregate's term
- The output-header variables are added as extra terms, so equal terms can still be counted multiple times

```
V3 = S :-
    S = #sum{ 1, C2 :
        V0 = yes,
        value(country_borders_country(C1, C2), V0) },
    value(nb_nbrs_of_country(C1), V3).
```

5. Data tables

Bordering countries			
D	Country called c1	Country called c2	c1 borders c2
1	Belgium	France, Luxembourg, Netherlands, Germany	Yes
2	Netherlands	Germany, Belgium	Yes
...	...	...	...

5. Data tables

Bordering countries			
D	Country called c1	Country called c2	c1 borders c2
1	Belgium	France, Luxembourg, Netherlands, Germany	Yes
2	Netherlands	Germany, Belgium	Yes
...	...	...	...

- Represent problem instance

```
input(country_borders_country(be, fr), yes).
input(country_borders_country(be, lu), yes).
input(country_borders_country(be, nl), yes).
...
input(country_borders_country(dk, de), yes).

input(country_borders_country(X0, X1), no) :-
    not input(country_borders_country(X0, X1), yes).

:- not value(X, Y), input(X, Y).
```

5. Data tables

Bordering countries			
D	Country called c1	Country called c2	c1 borders c2
1	Belgium	France, Luxembourg, Netherlands, Germany	Yes
2	Netherlands	Germany, Belgium	Yes
...	...	...	...

- Represent problem instance
- Totally define a symbol

```
input(country_borders_country(be, fr), yes).
input(country_borders_country(be, lu), yes).
input(country_borders_country(be, nl), yes).
...
input(country_borders_country(dk, de), yes).

input(country_borders_country(X0, X1), no) :-
    not input(country_borders_country(X0, X1), yes).

:- not value(X, Y), input(X, Y).
```

5. Data tables

Bordering countries			
D	Country called c1	Country called c2	c1 borders c2
1	Belgium	France, Luxembourg, Netherlands, Germany	Yes
2	Netherlands	Germany, Belgium	Yes
...	...	...	...

- Represent problem instance
- Totally define a symbol
- For predicates: implicit `no`

```
input(country_borders_country(be, fr), yes).
input(country_borders_country(be, lu), yes).
input(country_borders_country(be, nl), yes).
...
input(country_borders_country(dk, de), yes).

input(country_borders_country(X0, X1), no) :-
    not input(country_borders_country(X0, X1), yes).

:- not value(X, Y), input(X, Y).
```

Software

Before

- The **cDMN solver** (Python) translates cDMN → **FO(·)**
- FO(·) = an extended version of classical FOL
- Reasoners: **IDP3**, **IDP-Z3**, **SLI**

Software

Before

- The **cDMN solver** (Python) translates cDMN → **FO(·)**
- FO(·) = an extended version of classical FOL
- Reasoners: **IDP3**, **IDP-Z3**, **SLI**

Now

- A fully automatic translation **cDMN → ASP**

🔗 `gitlab.com/EAVISE/cdmn/cdmn-solver/-/tree/asp`

Software

Before

- The **cDMN solver** (Python) translates cDMN → **FO(·)**
- FO(·) = an extended version of classical FOL
- Reasoners: **IDP3**, **IDP-Z3**, **SLI**

Now

- A fully automatic translation **cDMN → ASP**

🔗 `gitlab.com/EAVISE/cdmn/cdmn-solver/-/tree/asp`

Why bother?

Software

Before

- The **cDMN solver** (Python) translates cDMN → **FO(·)**
- FO(·) = an extended version of classical FOL
- Reasoners: **IDP3**, **IDP-Z3**, **SLI**

Now

- A fully automatic translation **cDMN → ASP**

🔗 `gitlab.com/EAVISE/cdmn/cdmn-solver/-/tree/asp`

Why bother?

- For some problem classes, ASP solvers are simply **faster**
- Access to the ASP **ecosystem**:
  - `clinguin`, `ASP Chef`, …
  - `lpopt`, `ngo` (encoding optimisation)
- Opens the door to e.g. **multi-shot solving**

Software

Before

- The **cDMN solver** (Python) translates cDMN → **FO(·)**
- FO(·) = an extended version of classical FOL
- Reasoners: **IDP3**, **IDP-Z3**, **SLI**

Now

- A fully automatic translation **cDMN → ASP**

🔗 `gitlab.com/EAVISE/cdmn/cdmn-solver/-/tree/asp`

Why bother?

- For some problem classes, ASP solvers are simply **faster**
- Access to the ASP **ecosystem**:
  - `clinguin`, `ASP Chef`, …
  - `lpopt`, `ngo` (encoding optimisation)
- Opens the door to e.g. **multi-shot solving**

Status

Software

Before

- The **cDMN solver** (Python) translates cDMN → **FO(·)**
- FO(·) = an extended version of classical FOL
- Reasoners: **IDP3**, **IDP-Z3**, **SLI**

Now

- A fully automatic translation **cDMN → ASP**

🔗 `gitlab.com/EAVISE/cdmn/cdmn-solver/-/tree/asp`

Why bother?

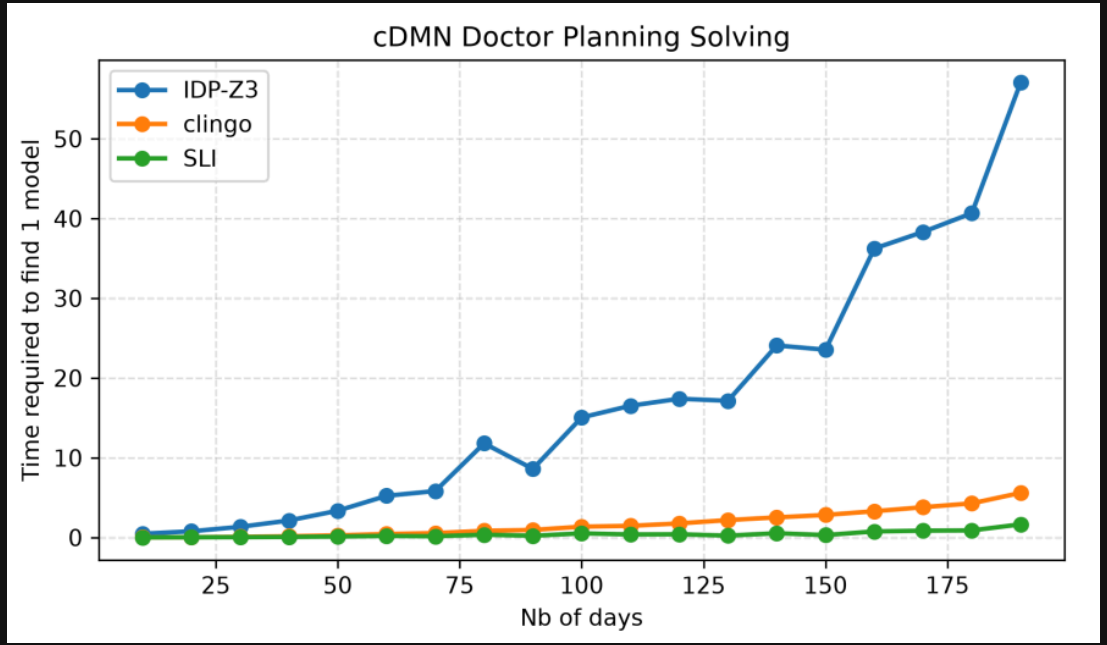
- For some problem classes, ASP solvers are simply **faster**
- Access to the ASP **ecosystem**:
  - `clinguin`, `ASP Chef`, …
  - `lpopt`, `ngo` (encoding optimisation)
- Opens the door to e.g. **multi-shot solving**

Status

- Translates many of the existing cDMN examples
- Models with `C+` tables are still slow — the translation is not optimal yet

Preliminary benchmark: Doctor Planning

Finding a permanence schedule for 5 doctors over *x* days (a simple rostering problem). All three systems use an encoding generated **directly by the cDMN solver**.



Thank you!

Questions?

`gitlab.com/EAVISE/cdmn/cdmn-solver/-/tree/asp`

Many thanks to Javier Romero for assistance with the initial translation!


https://simonvandevelde.be


s.vandevelde@kuleuven.be


@saltfactory@mastodon.social


Simon Vandevelde