cDMN: Combining DMN with Constraint Reasoning

DecisionCAMP2020

Simon Vandevelde, Bram Aerts,
Joost Vennekens
KU Leuven - EAVISE
30 June 2020

KU LEUVEN eavise

Outline

1. DMN
2. Constraint DMN
3. cDMN application: Doctor Planning
4. Comparison to other solutions
5. A user friendly pipeline from cDMN to interface

1 DMN

DMN

What DMN does well:

- Very readable
- Very intuitive
- Easy to formalize decision making

DMN

However: not well-suited for complex decision making:

- FEEL: can express a lot, but becomes unreadable
- No constraints
- No optimization

$\rightarrow$ we propose additions to overcome these limitations.

2 Constraint DMN

What is cDMN?

cDMN: constraint DMN

- Extends DMN with constraint reasoning
- Quantification, types, functions, relations, optimization, ..
- Model complex decisions
- Introduces new tables
  - Glossary
  - Constraint tables
  - Data tables
  - Execute tables

The Glossary

Changes in representation:

- DMN: constants
- cDMN: types, functions, relations, constants, booleans
  $\rightarrow$ defined by glossary

The Glossary

Changes in representation:

- DMN: constants
- cDMN: types, functions, relations, constants, booleans
  $\rightarrow$ defined by glossary

For example:

Type		
Name	**Type**	**Values**
Person	string	Agatha, Butler, Charles
Number	int	[0..100]

The Glossary

Changes in representation:

- DMN: constants
- cDMN: types, functions, relations, constants, booleans
  $\rightarrow$ defined by glossary

For example:

Type		
Name	**Type**	**Values**
Person	string	Agatha, Butler, Charles
Number	int	[0..100]

Relation
Name
Person hates Person
Person is richer than Person

The Glossary

Changes in representation:

- DMN: constants
- cDMN: types, functions, relations, constants, booleans
  $\rightarrow$ defined by glossary

For example:

Type		
Name	**Type**	**Values**
Person	string	Agatha, Butler, Charles
Number	int	[0..100]

Relation
Name
Person hates Person
Person is richer than Person

Boolean
Name
Suicide

The Glossary

Changes in representation:

- DMN: constants
- cDMN: types, functions, relations, constants, booleans
  $\rightarrow$ defined by glossary

For example:

Type		
Name	**Type**	**Values**
Person	string	Agatha, Butler, Charles
Number	int	[0..100]

Relation
Name
Person hates Person
Person is richer than Person

Boolean
Name
Suicide

Constant	
Name	**Type**
Killer	Person

The Glossary

Changes in representation:

- DMN: constants
- cDMN: types, functions, relations, constants, booleans
  $\rightarrow$ defined by glossary

For example:

Type		
Name	**Type**	**Values**
Person	string	Agatha, Butler, Charles
Number	int	[0..100]

Relation
Name
Person hates Person
Person is richer than Person

Boolean
Name
Suicide

Constant	
Name	**Type**
Killer	Person

Function	
Name	**Type**
Age of Person	Number
Hatees of Person	Number

Constraint tables

Constraint tables differ in two ways from decision tables:

1. Outputs can contain S-FEEL.
2. Rows express *logical implications*:
   - → IF input satisfied, THEN output has to be satisfied
   - → Nothing said about output when no input applicable (no Null)

E* hit policy

Constraint tables

Constraint tables differ in two ways from decision tables:

1. Outputs can contain S-FEEL.
2. Rows express *logical implications*:
   - → IF input satisfied, THEN output has to be satisfied
   - → Nothing said about output when no input applicable (no Null)

E* hit policy

Nutritional values				
E*	**RequireHealthy**	**Total Sodium**	**Total Fat**	**Total Calories**
1	Yes	<3000	<150	<3000

$$RequireHealthy \Rightarrow TotalSodium < 3000 \wedge TotalFat < 150 \wedge TotalCalories < 3000$$

Quantification

cDMN also allows *universal quantification*:

- Logic for all values of a type
- Type as column header
- Subsequent uses refer to the quantified variable.

Constraints and quantification: example

Example:

Noone hates all		
E*	**Person**	**Hatees of Person**
1	-	< 3

$$\forall Person[Person] : Hatees(Person) < 3$$

Constraints and quantification: example

Example:

Noone hates all		
E*	**Person**	**Hatees of Person**
1	-	< 3

$$\forall Person[Person] : Hatees(Person) < 3$$

Bordering countries can not share colors				
E*	**Country called c1**	**Country called c2**	**c1 and c2 are Bordering**	**Color of c1**
1	-	-	Yes	Not(Color of c2)

$$\forall c1[Country], c2[Country] : Bordering(c1, c2) \Rightarrow Color(c1) \neq Color(c2)$$

Data Tables

Typically, problems can be split up in two parts:

1. general logic of the problem
2. specific problem instance to solve

Data Tables

Typically, problems can be split up in two parts:

1. general logic of the problem
2. specific problem instance to solve

Map coloring problem:

1. Two bordering countries cannot share a color.
2. The specific map to color (e.g. Western Europe).

Data Tables

Logic: decision and constraint tables
Problem instance: data tables

Data Tables

Logic: decision and constraint tables
Problem instance: data tables

- No hit policy, but "Data Table" in name
- Only basic values
- Quantification possible

Data Tables

Logic: decision and constraint tables
Problem instance: data tables

- No hit policy, but "Data Table" in name
- Only basic values
- Quantification possible

Data Table: Declaring which countries border			
	Country called c1	**Country called c2**	**c1 and c2 are Bordering**
1	Belgium	France, Luxembourg, Netherlands, Germany	Yes
2	Germany	France, Denmark, Luxembourg, Belgium, Netherlands	Yes
	. . .	. . .	Yes

Model Solving and Optimization

- DMN always has a single solution for a set of inputs
- not the case in cDMN
  $\rightarrow$ cDMN defines solution space
- *Execute table* specifies solutions

Model Solving and Optimization

- DMN always has a single solution for a set of inputs
- not the case in cDMN
  $\rightarrow$ cDMN defines solution space
- *Execute table* specifies solutions

Execute
Get 1 model

Model Solving and Optimization

- DMN always has a single solution for a set of inputs
- not the case in cDMN
  $\rightarrow$ cDMN defines solution space
- *Execute table* specifies solutions

Execute
Get 1 model

Execute
Get all models

Model Solving and Optimization

- DMN always has a single solution for a set of inputs
- not the case in cDMN
  $\rightarrow$ cDMN defines solution space
- *Execute table* specifies solutions

Execute
Get 1 model

Execute
Get all models

Execute
Maximize Score

3 cDMN Application: Doctor Planning

Challenge Specification

Doctor Planning

- Doctor present at all times in a hospital
- Planning made per week
- Days consist of three shifts (early, late, night)
- Five doctors in total
- Each doctor has different availabilities

Challenge Specification: rules

Rules

1. A doctor can only work one shift per day.
2. A doctor should always be available for his shift (see table below).
3. If a doctor has the night shift, they either get the next day off, or the night shift again.
4. A doctor either works both days of the weekend, or none of the days.

Challenge Specification: Availabilities

Availabilities

Name	Available
Fleming	Friday, Saturday, Sunday
Freud	Every day early or late, never night
Heimlich	Every day but never the night shift on weekends
Eustachi	Every day, every shift
Golgi	Every day, every shift but at max 2 night shifts

A planning should be made in which every requirement is fullfilled.

Doctor Planning: Rules

Rule 1

A doctor can only work one shift per day.

Type		
Name	**Type**	**Values**
Doctor	string	Fleming, Freud, Heimlich, Eustachi, Golgi
Day	string	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Nb Shift	int	[0..21]

Function	
Name	**Type**
Nb Shifts of Doctor and Day	Nb Shift

Doctor Planning: Rules

Rule 1

A doctor can only work one shift per day.

Rule 1			
E*	**Doctor**	**Day**	**Nb Shifts of Doctor and Day**
1	-	-	≤ 1

Doctor Planning: Rules

Rule 2

A doctor should be available for his shift.

Type		
Name	**Type**	**Values**
Doctor	string	Fleming, Freud, Heimlich, Eustachi, Golgi
Day	string	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Nb Shift	int	[0..21]
Shift_Type	string	Early, Late, Night

Function	
Name	**Type**
Nb Shifts of Doctor and Day	Nb Shift
Assigned Doctor of Day and Shift_Type	Doctor

Relation
Name
Available Doctor for Shift_Type and Day

Doctor Planning: Rules

Rule 2

A doctor should be available for his shift.

Rule 2					
E*	**Doctor**	**Day**	**Shift_Type**	**Assigned Doctor of Day and Shift_Type**	**Available Doctor for Shift_Type and Day**
1	-	-	-	Doctor	Yes

Doctor Planning: Rules

Rule 3

If a doctor has the night shift, they either get the next day off, or the night shift again.

Type		
Name	**Type**	**Values**
Doctor	string	Fleming, Freud, Heimlich, Eustachi, Golgi
Day	string	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Nb Shift	int	[0..21]
Shift_Type	string	Early, Late, Night

Function	
Name	**Type**
Nb Shifts of Doctor and Day	Nb Shift
Assigned Doctor of Day and Shift_Type	Doctor
Next_Day of Day	Day

Relation
Name
Available Doctor for Shift_Type and Day

Doctor Planning: Rules

Rule 3

If a doctor has the night shift, they either get the next day off, or the night shift again.

Rule 3							
E*	**Doctor**	**Day called d1**	**Shift_Type called s1**	**Doctor of d1 and s1**	**Day called d2**	**Shift_Type called s2**	**Doctor of d2 and s2**
1	-	-	Night	Doctor	Next_Day of d1	Early, Late	not(Doctor)

Doctor Planning: Rules

Rule 4

A doctor either works both days of the weekend, or none of the days.

Type		
Name	**Type**	**Values**
Doctor	string	Fleming, Freud, Heimlich, Eustachi, Golgi
Day	string	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Nb Shift	int	[0..21]
Shift_Type	string	Early, Late, Night

Function	
Name	**Type**
Nb Shifts of Doctor and Day	Nb Shift
Assigned Doctor of Day and Shift_Type	Doctor
Next_Day of Day	Day

Relation
Name
Available Doctor for Shift_Type and Day

Doctor Planning: Rules

Rule 4

A doctor either works both days of the weekend, or none of the days.

Rule 4					
E*	**Doctor**	**Day called d1**	**Nb Shifts of Doctor and d1**	**Day called d2**	**Nb Shifts of Doctor and d2**
1	-	Saturday	1	Sunday	1
2	-	Sunday	1	Saturday	1

Doctor Planning: Rules

Special Preference

Golgi works at maximum two nights.

Type		
Name	**Type**	**Values**
Doctor	string	Fleming, Freud, Heimlich, Eustachi, Golgi
Day	string	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Nb Shift	int	[0..21]
Shift_Type	string	Early, Late, Night

Function	
Name	**Type**
Nb Shifts of Doctor and Day	Nb Shift
Assigned Doctor of Day and Shift_Type	Doctor
Next_Day of Day	Day
Nb Nights of Doctor	Nb Shift

Relation
Name
Available Doctor for Shift_Type and Day

Doctor Planning: Rules

Special Preference

Golgi works at maximum two nights.

Max Nights		
E*	**Doctor**	**Nb Nights of Doctor**
1	Golgi	≤ 2

Doctor Planning: Availabilities

Data Table: Availabilities				
	Doctor	**Shift_Type**	**Day_Type**	**Available Doctor for Shift_Type and Day**
1	Fleming	Early	Friday	Yes
2	Fleming	Late	Friday	Yes
3	Fleming	Night	Friday	Yes
4	Fleming	Early	Saturday	Yes
	. . .	. . .	. . .	. . .

Doctor Planning: other tables

Next Day		
U	**Day**	**Next_Day of Day**
1	Sunday	Monday
2	Monday	Tuesday
3	Tuesday	Wednesday
4	Wednesday	Thursday
5	Thursday	Friday
6	Friday	Saturday
7	Saturday	Sunday

Doctor Planning: other tables

Next Day		
U	**Day**	**Next_Day of Day**
1	Sunday	Monday
2	Monday	Tuesday
3	Tuesday	Wednesday
4	Wednesday	Thursday
5	Thursday	Friday
6	Friday	Saturday
7	Saturday	Sunday

Count shifts per day					
C+	**Doctor**	**Day**	**Shift_Type**	**Assigned Doctor of Day and Shift_Type**	**Nb Shifts of Doctor and Day**
1	-	-	-	Doctor	1

Doctor Planning: other tables

Next Day

U	Day	Next_Day of Day
1	Sunday	Monday
2	Monday	Tuesday
3	Tuesday	Wednesday
4	Wednesday	Thursday
5	Thursday	Friday
6	Friday	Saturday
7	Saturday	Sunday

Count shifts per day

C+	Doctor	Day	Shift_Type	Assigned Doctor of Day and Shift_Type	Nb Shifts of Doctor and Day
1	-	-	-	Doctor	1

Count night shifts for every doctor

C+	Doctor	Day	Shift_Type	Assigned Doctor of Day and Shift_Type	Nb Nights of Doctor
1	-	-	Night	Doctor	1

Doctor Planning: Full

Define Next Day

U	Day	Next_Day of Day
1	Monday	Tuesday
2	Tuesday	Wednesday
3	Wednesday	Thursday
4	Thursday	Friday
5	Friday	Saturday
6	Saturday	Sunday
7	Sunday	Monday

Type

Name	DataType	Possible Values
Doctor	string	Fleming, Freud, Heimlich, Eustachi, Golgi
Shift_Type	string	Early, Late, Night
Day	string	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Nb Shift	int	[0..21]

Function

Name	DataType
Assigned Doctor of Day and Shift_Type	Doctor
Day of Shift	Day
Nb Shifts of Doctor and Day	Nb Shift
Nb Nights of Doctor	Nb Shift
Next_Day of Day	Day

Relation

Name
Available Doctor for Shift_Type and Day

Count shifts per day

C+	Doctor	Day	Shift_Type	Assigned Doctor of Day and Shift_Type	Nb Shifts of Doctor and Day
1	-	-	-	Doctor	1

1. A doctor can only work one shift per d

E*	Doctor	Day	Nb Shifts of Doctor and Day
1	-	-	<= 1

2. A doctor should always be available for his shift.

E*	Doctor	Day	Shift_Type	Assigned Doctor of Day and Shift_Type	Available Doctor for Shift_Type and Day
1	-	-	-	Doctor	Yes

3. If a doctor has the night shift, they either get the next day off, or the night shift again.

E*	Doctor	Day called d1	Shift_Type called s1	Assigned Doctor of d1 and s1	Day called d2	Shift_Type called s2	Assigned Doctor of d2 and s2
1	-	-	Night	Doctor	Next_Day of d1	not(Night)	not(Doctor)

4. A doctor either works both days of the weekend, or none of the days.

E*	Doctor	Day called d1	Nb Shifts of Doctor and d1	Day called d2	Nb Shifts of Doctor and d2
1	-	Saturday	1	Sunday	1
2	-	Sunday	1	Saturday	1

Max Nb of Nights

E*	Doctor	Nb Nights of Doctor
1	Golgi	<= 2

Count the number of night shifts for each doctor

C+	Doctor	Day	Shift_Type	Assigned Doctor of Day and Shift_Type	Nb Nights of Doctor
1	-	-	Night	Doctor	1

Data Table: available doctors

	Doctor	Shift_Type	Day	Available Doctor for Shift_Type and Day
1	Fleming	Early	Friday	Yes
2	Fleming	Late	Friday	Yes
3	Fleming	Night	Friday	Yes
4	Fleming	Early	Saturday	Yes
5				

4 Comparison to other solutions

Comparison

On 10th of June, these were the solutions:

- OPL (Alex Fleisher)
- PostgreSQL (Damir Sudarevic)
- ZIMPL (Rob Parker)
- Prolog (Matteo Redaelli)
- OpenRules, JavaSolver, AWS Lambda (Jacob Feldman)

None are DMN-like implementations.

Comparison

OPL:

```
// night shift ==> next day off or next night shift

forall(d in doctors,i in days:(i+1) in days)
 {
   (x[d][i]["night"]==1) => (x[d][i+1]["early"]==0);
   (x[d][i]["night"]==1) => (x[d][i+1]["late"]==0);
}
```

COMPARISON

ZIMPL:

```
# If a doctor works night shift, they cannot work early or late the next day...

subto Night_Shift_Continuity1:

forall <p> in PROFESSIONALS :

  forall <d> in DAYS with d < 7:

    vif (ASSIGNED[d,"Night",p] >= 1) then

        ASSIGNED[d+1,"Early",p] <=0

    end;


subto Night_Shift_Continuity2:
```

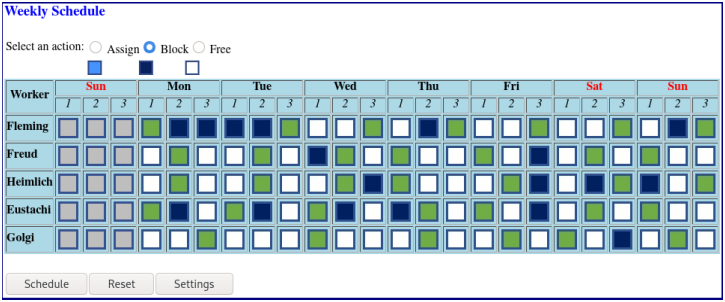
Comparison

OpenRules, JavaSolver, AWS Lambda:

```
// If a doctor has the night shift,
// they either get next day off or the the night shift again
if (day > 0) {
    if (shift < nightShift && !Doctor.FROZEN.equals(status)) {
        Var previousNightShift = vars[doc][day - 1][nightShift];
        Constraint constraint1 = csp.linear(previousNightShift, "=", 1);
        Constraint constraint2 = csp.linear(var, "=", 0);
        csp.postIfThen(constraint1, constraint2);
    }
}
```

COMPARISON

OpenRules, JavaSolver, AWS Lambda:



Live Worker Scheduler

Jacob Feldman created a full-blown planning application

- The GUI is created using OpenRules Dialog
- The scheduling is implemented in JavaSolver
+ The GUI is very intuitive and the system works well
- The solver is written in Java, only accessible to software developers

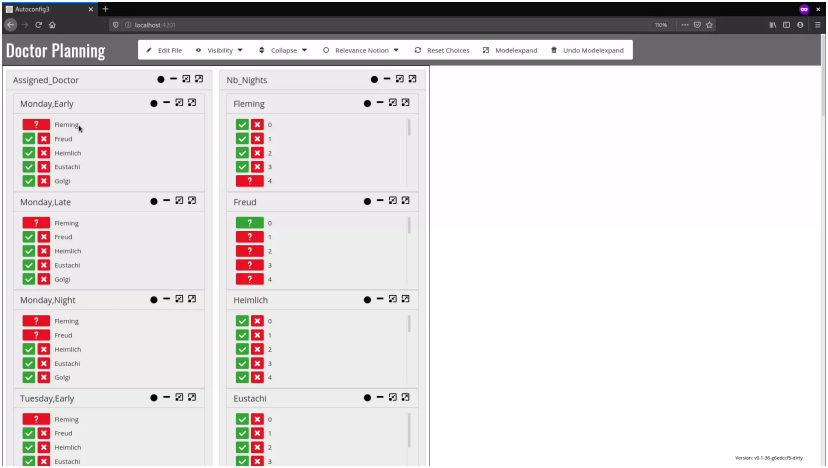
5 A USER FRIENDLY PIPELINE FROM cDMN TO INTERFACE

Autoconfig Interface

Using the cDMN solver, a cDMN specification can be fed to the Autoconfig interface

- Users can interact with the cDMN specifications
- Users can set values, and see the results
- The system doesn't distinguish between inputs or outputs
- The system can find optimal solutions

Autoconfig Interface Demo



Thank you for your attention.

Any questions?

For further questions or discussion:
s.vandevelde@kuleuven.be

For more information on cDMN and the cDMN solver:
www.cdmn.be

For more information on the interface:
https://gitlab.com/krr/autoconfigz3