# Efficient ASP Techniques for Solving Hard Problems

Carmine Dodaro
DIBRIS, University of Genoa

Arcavacata di Rende, 2-5-6-7 February 2018

# Previous Lesson

- **The idea of ASP**:
  1. Write a program representing a computational problem
     $\rightarrow$ i.e., such that answer sets correspond to solutions
  2. Use a solver to find solutions

# Previous Lesson

- **The idea of ASP**:
  1. Write a program representing a computational problem
     $\rightarrow$ i.e., such that answer sets correspond to solutions
  2. Use a solver to find solutions

- **Why is the knowledge of ASP Solving important?**

# Previous Lesson

- **The idea of ASP**:
  1. Write a program representing a computational problem
     $\rightarrow$ i.e., such that answer sets correspond to solutions
  2. Use a solver to find solutions

- **Why is the knowledge of ASP Solving important?**
  - Knowledge of programming methodology
    $\rightarrow$ you can write programs

# Previous Lesson

- **The idea of ASP**:
  1. Write a program representing a computational problem
     $\rightarrow$ i.e., such that answer sets correspond to solutions
  2. Use a solver to find solutions

- **Why is the knowledge of ASP Solving important?**
  - Knowledge of programming methodology
    $\rightarrow$ you can write programs
  - Knowledge of the evaluation process
    $\rightarrow$you can write programs more efficiently

# Previous Lesson

- **The idea of ASP**:
  1. Write a program representing a computational problem
     $\rightarrow$ i.e., such that answer sets correspond to solutions
  2. Use a solver to find solutions

- **Why is the knowledge of ASP Solving important?**
  - Knowledge of programming methodology
    $\rightarrow$ you can write programs
  - Knowledge of the evaluation process
    $\rightarrow$you can write programs more efficiently
  - Knowledge of an ASP System
    $\rightarrow$ you can actually implement applications and extension

# Evaluation of ASP Programs

- Computationally expensive
- Traditionally a two-step process:
  1. Instantiation (or grounding)

     $\rightarrow$ Variable elimination

  2. Propositional search (or solving)

     $\rightarrow$ Produce answer sets

# About the Instantiation

**Some facts:**

- Exponential in the worst case
- Input of a subsequent exponential procedure
- Significantly affects the performance of the overall process

# About the Instantiation

**Some facts:**

- Exponential in the worst case
- Input of a subsequent exponential procedure
- Significantly affects the performance of the overall process

**Intelligent instantiation**

- Keep the size of the instantiation as small as possible
- grounders can solve polynomial problems

# About the Instantiation

**Some facts:**

- Exponential in the worst case
- Input of a subsequent exponential procedure
- Significantly affects the performance of the overall process

**Intelligent instantiation**

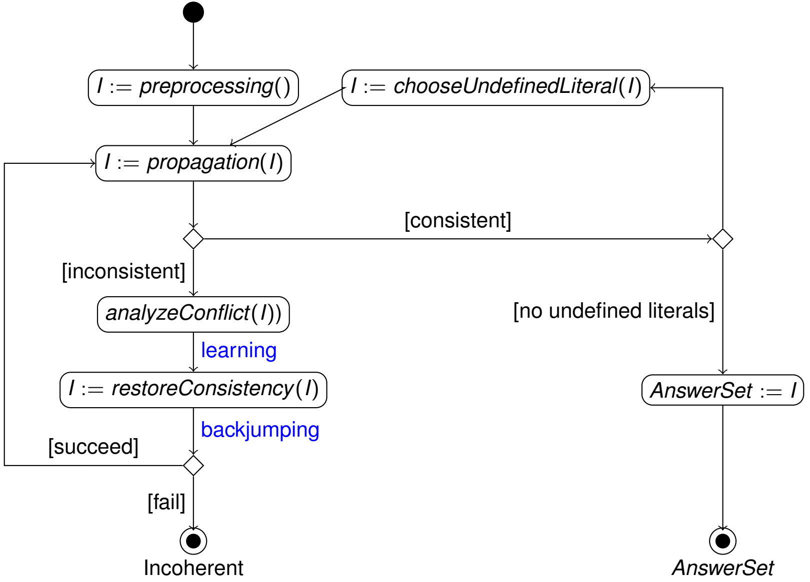
- Keep the size of the instantiation as small as possible
- grounders can solve polynomial problems

N.B: Naive encodings can lead to the grounding bottleneck

# Solver

- The input is a variable-free ASP program
- The theoretical search space is $O(2^n)$, where $n$ is the number of atoms
- Produces (optimum) answer sets
  - Techniques from SAT
  - Backtracking search
  - Based on the pattern: *Choose* – *Propagate* – *Learn*

# Solver: The Algorithm



# Propagation

## Derivation Rules

1. Unit propagation (from SAT)
2. Aggregates propagation (from Pseudo-Boolean)
3. Unfounded-free propagation (ASP specific)

# Unit and Aggregate propagation

- Infer a literal if it is the only one which can satisfy a rule

**Example (Unit propagation)**

```
a :- b, c.
```

If *b* and *c* are true then *a* must be true

- Uses aggregates for further inferences

**Example (Aggregate propagation)**

```
:- #sum{1,d : d; 2,e : e; 1,f : f} >= 2
```

If *d* is true then *e* and *f* must be false

# Unfounded-free propagation

- All atoms in an unfounded set are inferred as false

**Example (Unfounded set)**

```
a :- b
b :- a
```

$\{a, b\}$ is an unfounded set, thus *a* and *b* are inferred as false

# Solver: An Example

**Solver step:**

$$col(1, red) \mid col(1, yellow) \mid col(1, green).$$
$$col(2, red) \mid col(2, yellow) \mid col(2, green).$$
$$col(3, red) \mid col(3, yellow) \mid col(3, green).$$

$$:\!-\ col(1, red),\ col(2, red).$$
$$:\!-\ col(1, green),\ col(2, green).$$
$$:\!-\ col(1, yellow),\ col(2, yellow).$$
$$:\!-\ col(2, red),\ col(3, red).$$
$$:\!-\ col(2, green),\ col(3, green).$$
$$:\!-\ col(2, yellow),\ col(3, yellow).$$

Idea: Build an answer set step by step

**True:** {}

**False:** {}

# Solver: An Example

**Solver step: Choose literal**

$$col(1, red) \mid col(1, yellow) \mid col(1, green).$$
$$col(2, red) \mid col(2, yellow) \mid col(2, green).$$
$$col(3, red) \mid col(3, yellow) \mid col(3, green).$$

$$:\!-\ col(1, red),\ col(2, red).$$
$$:\!-\ col(1, green),\ col(2, green).$$
$$:\!-\ col(1, yellow),\ col(2, yellow).$$
$$:\!-\ col(2, red),\ col(3, red).$$
$$:\!-\ col(2, green),\ col(3, green).$$
$$:\!-\ col(2, yellow),\ col(3, yellow).$$

**True:** $\{\} \leftarrow col(1, red)$
**False:** $\{\}$

# Solver: An Example

**Solver step:** **Propagate Deterministic Consequences**

$col(1, red) \mid col(1, yellow) \mid col(1, green).$ $\leftarrow$ 1-minimality propagation
$col(2, red) \mid col(2, yellow) \mid col(2, green).$
$col(3, red) \mid col(3, yellow) \mid col(3, green).$

$:\!-\ col(1, red),\ col(2, red).$ $\leftarrow$ 2-unit propagation
$:\!-\ col(1, green),\ col(2, green).$
$:\!-\ col(1, yellow),\ col(2, yellow).$
$:\!-\ col(2, red),\ col(3, red).$
$:\!-\ col(2, green),\ col(3, green).$
$:\!-\ col(2, yellow),\ col(3, yellow).$

**True:** $\{col(1, red)\}$
**False:** $\{col(1, yellow),\ col(1, green),\ col(2, red)\}$

# Solver: An Example

**Solver step: Choose literal**

$$col(1, red) \mid col(1, yellow) \mid col(1, green).$$
$$col(2, red) \mid col(2, yellow) \mid col(2, green).$$
$$col(3, red) \mid col(3, yellow) \mid col(3, green).$$

$$:- \ col(1, red), \ col(2, red).$$
$$:- \ col(1, green), \ col(2, green).$$
$$:- \ col(1, yellow), \ col(2, yellow).$$
$$:- \ col(2, red), \ col(3, red).$$
$$:- \ col(2, green), \ col(3, green).$$
$$:- \ col(2, yellow), \ col(3, yellow).$$

**True:** $\{col(1, red)\} \leftarrow col(2, yellow)$
**False:** $\{col(1, yellow), \ col(1, green), \ col(2, red)\}$

# Solver: An Example

**Solver step:** **Propagate Deterministic Consequences**

$col(1, red) \mid col(1, yellow) \mid col(1, green).$
$col(2, red) \mid col(2, yellow) \mid col(2, green).$ ←1-minimality propagation
$col(3, red) \mid col(3, yellow) \mid col(3, green).$

$:-\ col(1, red),\ col(2, red).$
$:-\ col(1, green),\ col(2, green).$
$:-\ col(1, yellow),\ col(2, yellow).$
$:-\ col(2, red),\ col(3, red).$
$:-\ col(2, green),\ col(3, green).$
$:-\ col(2, yellow),\ col(3, yellow).$ ← 2-unit propagation

**True:** $\{col(1, red), col(2, yellow)\}$
**False:** $\{col(1, yellow),\ col(1, green),\ col(2, red), col(2, green), col(3, yellow)\}$

# Solver: An Example

**Solver step:** **Choose literal**

$$\mathit{col}(1, \mathit{red}) \mid \mathit{col}(1, \mathit{yellow}) \mid \mathit{col}(1, \mathit{green}).$$
$$\mathit{col}(2, \mathit{red}) \mid \mathit{col}(2, \mathit{yellow}) \mid \mathit{col}(2, \mathit{green}).$$
$$\mathit{col}(3, \mathit{red}) \mid \mathit{col}(3, \mathit{yellow}) \mid \mathit{col}(3, \mathit{green}).$$

$$:\!-\ \mathit{col}(1, \mathit{red}),\ \mathit{col}(2, \mathit{red}).$$
$$:\!-\ \mathit{col}(1, \mathit{green}),\ \mathit{col}(2, \mathit{green}).$$
$$:\!-\ \mathit{col}(1, \mathit{yellow}),\ \mathit{col}(2, \mathit{yellow}).$$
$$:\!-\ \mathit{col}(2, \mathit{red}),\ \mathit{col}(3, \mathit{red}).$$
$$:\!-\ \mathit{col}(2, \mathit{green}),\ \mathit{col}(3, \mathit{green}).$$
$$:\!-\ \mathit{col}(2, \mathit{yellow}),\ \mathit{col}(3, \mathit{yellow}).$$

**True:** $\{\mathit{col}(1, \mathit{red}), \mathit{col}(2, \mathit{yellow})\} \leftarrow \mathit{col}(3, \mathit{red})$
**False:** $\{\mathit{col}(1, \mathit{yellow}),\ \mathit{col}(1, \mathit{green}),\ \mathit{col}(2, \mathit{red}),\ \mathit{col}(2, \mathit{green}),$
$\mathit{col}(3, \mathit{yellow})\}$

# Solver: An Example

**Solver step:** **Propagate Deterministic Consequences**

$col(1, red) \mid col(1, yellow) \mid col(1, green).$
$col(2, red) \mid col(2, yellow) \mid col(2, green).$
$col(3, red) \mid col(3, yellow) \mid col(3, green).$ ←minimality propagation

$:-\ col(1, red),\ col(2, red).$
$:-\ col(1, green),\ col(2, green).$
$:-\ col(1, yellow),\ col(2, yellow).$
$:-\ col(2, red),\ col(3, red).$
$:-\ col(2, green),\ col(3, green).$
$:-\ col(2, yellow),\ col(3, yellow).$

**True:** $\{col(1, red), col(2, yellow), col(3, red)\}$
**False:** $\{col(1, yellow),\ col(1, green),\ col(2, red),\ col(2, green),$
$col(3, yellow)\ col(3, green)\}$

# Solver: An Example

**Solver step:** Answer set found!

$$col(1, red) \mid col(1, yellow) \mid col(1, green).$$
$$col(2, red) \mid col(2, yellow) \mid col(2, green).$$
$$col(3, red) \mid col(3, yellow) \mid col(3, green).$$

$$:\!-\ col(1, red),\ col(2, red).$$
$$:\!-\ col(1, green),\ col(2, green).$$
$$:\!-\ col(1, yellow),\ col(2, yellow).$$
$$:\!-\ col(2, red),\ col(3, red).$$
$$:\!-\ col(2, green),\ col(3, green).$$
$$:\!-\ col(2, yellow),\ col(3, yellow).$$

**Answer Set:** $\{col(1, red), col(2, yellow), col(3, red)\}$

# Heuristics and learning

**Learning**

- Detect the reason of a conflict
- Learn constraints using 1-UIP schema

**Deletion Policy**

- Exponentially many constraints $\rightarrow$ forget something
- Less "useful" constraints are removed

**Search Restarts**

- Avoid unfruitful branches by restarting the search
- Based on some heuristic sequence

**Branching Heuristics**

- Look back MINISAT heuristic

# Optimum answer set search

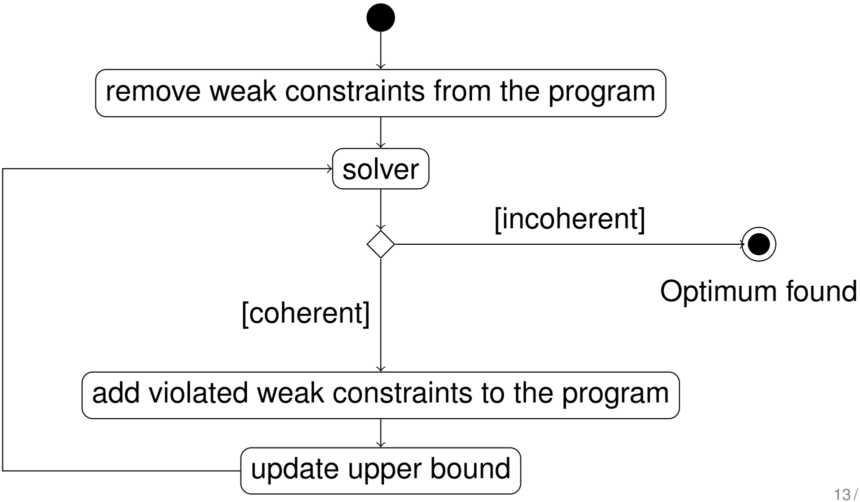
- What about programs with weak constraints?
- **Find the answer set with the minimum cost**
  - **Input:** a propositional program $\Pi$
  - **Output:** an optimum answer set of $\Pi$
- Based on MaxSAT algorithms
  - Model-guided
  - Core-guided

# Optimum answer set search

- Model-guided algorithms: OPT, BASIC and MGD
  - + Easy to implement
  - + Work well on particular domains
  - + Produce feasible solutions during the search
  - - Poor performances on industrial instances
- Core-guided algorithms: PMRES and OLL
  - + Good performances on industrial instances
  - - Do not produce feasible solutions (in general)
  - - The implementation is usually nontrivial

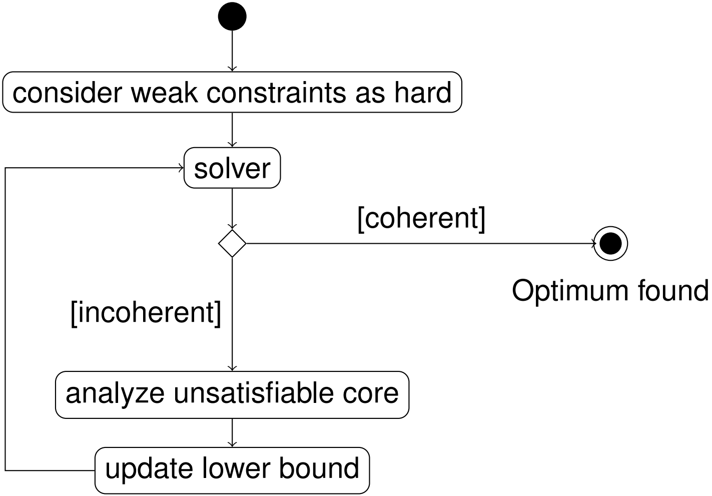
# Model-guided algorithms

I need a solution! Give me any answer set



# Core-guided algorithms

I feel lucky! Try to satisfy all weak constraints



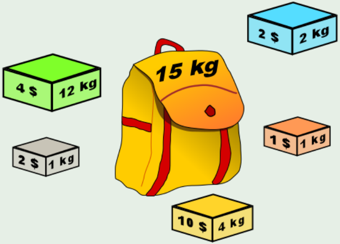
# Optimization problems in ASP

## Example (Knapsack)

- Stole as much value as possible

```
{in(X)}:– object(X).
:– #sum{W,X : weight(X,W), in(X)} > 15.
:∼ value(X,V), not in(X). [V@1,X]

object(green). ...
value(green,4). ...
weight(green,12). ...
```



# Model-guided: Solve by adding objects



a possible solution

# Model-guided: Solve by adding objects



a possible solution


better value

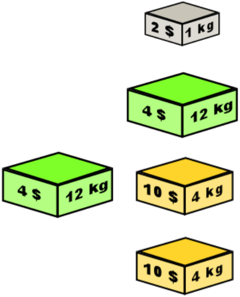


a possible solution


better value


not acceptable weight

# Model-guided: Solve by adding objects

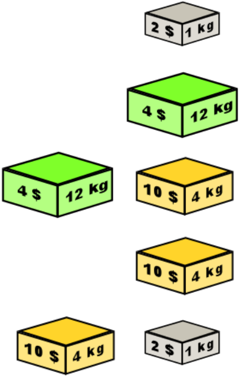

a possible solution

better value

not acceptable weight

better value

# Model-guided: Solve by adding objects


a possible solution

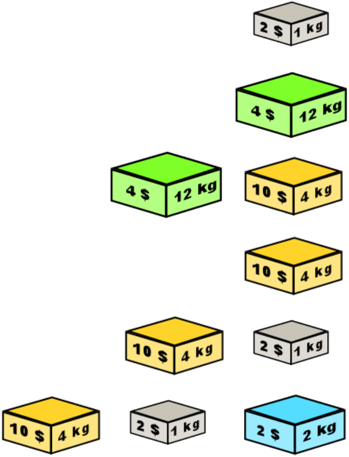
better value

not acceptable weight

better value

better value

# Model-guided: Solve by adding objects


a possible solution

better value

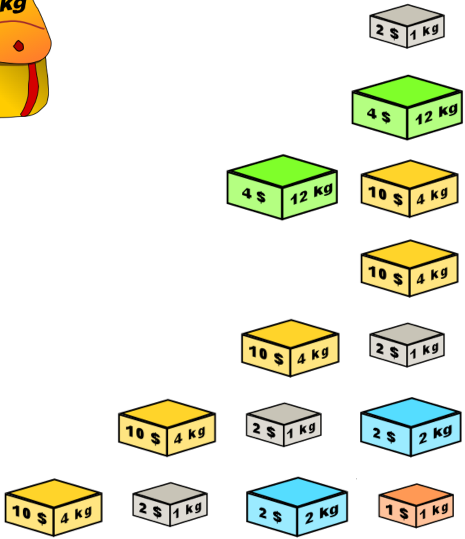
not acceptable weight

better value

better value

better value

# Model-guided: Solve by adding objects


a possible solution

better value

not acceptable weight

better value

better value

better value

better value (optimum)

# Core-guided: Solve by removing objects


Try

# Core-guided: Solve by removing objects


Try


Incompatibility


(an unsatisfiable core)

# Core-guided: Solve by removing objects


Try


Incompatibility


(an unsatisfiable core)

Replace by


where


implies one of

# Core-guided: Solve by removing objects


Try 


Incompatibility 

(an unsatisfiable core)

Replace by 


where 

implies one of 


Try 

# Core-guided: Solve by removing objects


Try 


Incompatibility 

(an unsatisfiable core)

Replace by 


where 

implies one of 


Try 


We have an optimum solution

# Programming for performance: basic idea

## Example (Maximal Clique)

**Problem:** Given an indirected Graph compute a clique of maximal size
**Input:** *node*(_) and *edge*(_, _).

# Programming for performance: basic idea

## Example (Maximal Clique)

**Problem:** Given an indirected Graph compute a clique of maximal size
**Input:** *node*(_) and *edge*(_, _).

**Natural Encoding:**

$inClique(X) \mid outClique(X) \text{ :- } node(X).$ % Guess

$\text{:- } inClique(X), inClique(Y), not\ edge(X, Y), X <> Y.$ % Check

$:\sim outClique(X).[1@1, X]$ % Optimize

# Programming for performance: basic idea

## Example (Maximal Clique)

**Problem:** Given an indirected Graph compute a clique of maximal size
**Input:** *node*(_) and *edge*(_, _).

**Natural Encoding:**

$inClique(X) \mid outClique(X) \text{ :- } node(X).$ % Guess
$\text{:- } inClique(X), inClique(Y), not\ edge(X, Y), X <> Y.$ % Check
$:\sim outClique(X).[1@1, X]$ % Optimize

**First Optimization:**

$inClique(X) \mid outClique(X) \text{ :- } node(X).$
$\text{:- } inClique(X), inClique(Y), not\ edge(X, Y), X < Y.$ $\leftarrow$ less constraints!
$:\sim outClique(X).[1@1, X]$

# Programming for performance: basic idea

## Example (Maximal Clique)

**Problem:** Given an indirected Graph compute a clique of maximal size
**Input:** *node*(_) and *edge*(_, _).

**Natural Encoding:**

$inClique(X) \mid outClique(X) \text{ :- } node(X).$ % Guess
$\text{:- } inClique(X), inClique(Y), not\ edge(X, Y), X <> Y.$ % Check
$:\sim outClique(X).[1@1, X]$ % Optimize

**First Optimization:**

$inClique(X) \mid outClique(X) \text{ :- } node(X).$
$\text{:- } inClique(X), inClique(Y), not\ edge(X, Y), X < Y.$ ← less constraints!
$:\sim outClique(X).[1@1, X]$

**Second Optimization:**

$\{inClique(X)\} \text{ :- } node(X).$
$\text{:- } inClique(X), inClique(Y), not\ edge(X, Y), X < Y.$
$:\sim node(X), not\ inClique(X).[1@1, X]$ ← removed outClique!

# Impact of Optimizations

- How many constraints are not used in the optimized encoding?
- What is the theoretical search space of the two encodings?
- Consider a complete graph with 50 nodes

# Impact of Optimizations

- How many constraints are not used in the optimized encoding?
- What is the theoretical search space of the two encodings?
- Consider a complete graph with 50 nodes
  - Natural encoding: 2450 constraints

# Impact of Optimizations

- How many constraints are not used in the optimized encoding?
- What is the theoretical search space of the two encodings?
- Consider a complete graph with 50 nodes
  - Natural encoding: 2450 constraints
  - Optimized encoding: 1225 constraints

## Impact of Optimizations

- How many constraints are not used in the optimized encoding?
- What is the theoretical search space of the two encodings?
- Consider a complete graph with 50 nodes
  - Natural encoding: 2450 constraints
  - Optimized encoding: 1225 constraints
  - Natural encoding: $2^{100}$ (50 atoms of type inClique and 50 atoms of the type outClique)

# Impact of Optimizations

- How many constraints are not used in the optimized encoding?
- What is the theoretical search space of the two encodings?
- Consider a complete graph with 50 nodes
  - Natural encoding: 2450 constraints
  - Optimized encoding: 1225 constraints
  - Natural encoding: $2^{100}$ (50 atoms of type inClique and 50 atoms of the type outClique)
  - Optimized encoding: $2^{50}$ (50 atoms of type inClique)

# Impact of Optimizations

- How many constraints are not used in the optimized encoding?
- What is the theoretical search space of the two encodings?
- Consider a complete graph with 50 nodes
  - Natural encoding: 2450 constraints
  - Optimized encoding: 1225 constraints
  - Natural encoding: $2^{100}$ (50 atoms of type inClique and 50 atoms of the type outClique)
  - Optimized encoding: $2^{50}$ (50 atoms of type inClique)

# Programming for performance: basic idea (2)

## Example (3-col- encoding 1)

% guess a coloring for the nodes

$col(X, red) \mid col(X, yellow) \mid col(X, blue) \ :\!-\ node(X).$

% check condition

$:\!-\ edge(X, Y), col(X, C),\ col(Y, C).$

## Example (3-col- encoding 2 )

% guess a coloring for the nodes

```
col(X,red)     |     ncol(X,red)     :- node(X).    ← three times
col(X,yellow)  |  ncol(X,yellow)     :- node(X).    ← more
col(X,blue)    |    ncol(X,blue)     :- node(X).    ← ground rules
```

% check condition

$:\!-\ edge(X, Y), col(X, C),\ col(Y, C).$

$:\!-\ col(X, C1),\ col(Y, C2), C1 <> C2.$

# Programming for performance: basic idea (2)

## Example (3-col- encoding 1)

% guess a coloring for the nodes

$col(X, red) \mid col(X, yellow) \mid col(X, blue) \;\text{:-}\; node(X).$

% check condition

$\text{:-}\; edge(X, Y), col(X, C),\; col(Y, C).$

% NB: answer sets are subset minimal $\rightarrow$ only one color per node

## Example (3-col- encoding 2 )

% guess a coloring for the nodes

```
col(X,red)     |     ncol(X,red)     :- node(X).
col(X,yellow)  |  ncol(X,yellow)     :- node(X).
col(X,blue)    |    ncol(X,blue)     :- node(X).
```

$\leftarrow$ three times
$\leftarrow$ more
$\leftarrow$ ground rules

% check condition

$\text{:-}\; edge(X, Y), col(X, C),\; col(Y, C).$

$\text{:-}\; col(X, C1),\; col(Y, C2), C1 <> C2.$ $\leftarrow$ additional constraint

# Programming for performance: basic idea (2)

## Example (3-col- encoding 1)

% guess a coloring for the nodes

$col(X, red) \mid col(X, yellow) \mid col(X, blue) \ \text{:-}\ node(X).$

% check condition

$\text{:-}\ edge(X, Y), col(X, C),\ col(Y, C).$

## Example (3-col- encoding 2 - **Larger grounding!** )

% guess a coloring for the nodes

```
col(X,red)     |    ncol(X,red)     :- node(X).   ← three times
col(X,yellow)  |    ncol(X,yellow)  :- node(X).   ← more
col(X,blue)    |    ncol(X,blue)    :- node(X).   ← ground rules
```

% check condition

$\text{:-}\ edge(X, Y), col(X, C),\ col(Y, C).$

$\text{:-}\ col(X, C1),\ col(Y, C2), C1 <> C2.$

# Programming for performance: basic idea (2)

## Example (3-col- encoding 1)

% guess a coloring for the nodes

$col(X, red) \mid col(X, yellow) \mid col(X, blue) \ \text{:-}\ node(X).$

% check condition

$\text{:-}\ edge(X, Y), col(X, C),\ col(Y, C).$

## Example (3-col- encoding 2 - **Larger Search Space!** )

% guess a coloring for the nodes

| | | | | | |
|---|---|---|---|---|---|
| col(X,red) | \| | ncol(X,red) | :- | node(X). | ← three times |
| col(X,yellow) | \| | ncol(X,yellow) | :- | node(X). | ← more |
| col(X,blue) | \| | ncol(X,blue) | :- | node(X). | ← ground rules |

% check condition

$\text{:-}\ edge(X, Y), col(X, C),\ col(Y, C).$

$\text{:-}\ col(X, C1),\ col(Y, C2), C1 <> C2.$

# Programming for performance: lesson learned

**Prefer an encoding if:**

- Easier to ground

  $\rightarrow$ precomputes as much as possible
- Smaller instantiation

  $\rightarrow$ use e.g., minimality, aggregates, ...
- Produces less ground disjunctive rules and less “guessed atoms”

  $\rightarrow$ smaller search space

  $\rightarrow$ exponential gain

# Programming Hints

**Programming for Performance:**

1. Consider complexity issues
2. Prefer Smaller/Faster Grounding
3. Reduce Search Space

# Programming Hints

**Programming for Performance:**

1. Consider complexity issues
2. Prefer Smaller/Faster Grounding
3. Reduce Search Space
4. Exploit the features of the implementation

# Grounding Bottleneck

- When the time and/or the memory required to compute the instantiation is too huge
- When the number of produced rules cannot be processed by the solver

# Grounding Bottleneck

- When the time and/or the memory required to compute the instantiation is too huge
- When the number of produced rules cannot be processed by the solver

Possible solutions?

- Improve the encoding!

# Grounding Bottleneck

- When the time and/or the memory required to compute the instantiation is too huge
- When the number of produced rules cannot be processed by the solver

Possible solutions?

- Improve the encoding!
- Use approaches based on lazy grounding

# Grounding Bottleneck

- When the time and/or the memory required to compute the instantiation is too huge
- When the number of produced rules cannot be processed by the solver

Possible solutions?

- Improve the encoding!
- Use approaches based on lazy grounding
- Replace portion of the encoding using dedicated propagators

# Grounding Bottleneck

- When the time and/or the memory required to compute the instantiation is too huge
- When the number of produced rules cannot be processed by the solver

Possible solutions?

- Improve the encoding!
- Use approaches based on lazy grounding
- Replace portion of the encoding using dedicated propagators

# Stable Marriage

## Definition

Given $n$ men and $n$ women, where each person has ranked all members of the opposite sex with a unique number between 1 and $n$ in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners.

| M | W |
|---|---|
| john | mary |
| luca | anna |

| P1 | P2 | Pref |
|---|---|---|
| john | mary | 1 |
| john | anna | 2 |
| luca | mary | 2 |
| luca | anna | 1 |

| P1 | P2 | Pref |
|---|---|---|
| mary | john | 1 |
| anna | john | 2 |
| mary | luca | 2 |
| anna | luca | 1 |

# Stable Marriage: Natural Encoding

```
% guess matching
match(M,W) | nMatch(M,W) :- man(M), woman(W).

% no polygamy
:- match(M1,W), match(M,W), M <> M1.
:- match(M,W), match(M,W1), W <> W1.

% no singles
married(M) :- match(M,W).
:- man(M), not married(M).

% strong stability condition
:- match(M,W1), match(M1,W), W1 <> W,
   pref(M,W,Smw), pref(M,W1,Smw1), Smw > Smw1,
   pref(W,M,Swm), pref(W,M1,Swm1), Swm >= Swm1.
```

# Stable Marriage: First Optimization

```
% guess matching
{match(M,W)} :- man(M), woman(W).

% no polygamy
:- match(M1,W), match(M,W), M <> M1.
:- match(M,W), match(M,W1), W <> W1.

% no singles
married(M) :- match(M,W).
:- man(M), not married(M).

% strong stability condition
:- match(M,W1), match(M1,W), W1 <> W,
   pref(M,W,Smw), pref(M,W1,Smw1), Smw > Smw1,
   pref(W,M,Swm), pref(W,M1,Swm1), Swm >= Swm1.
```

# Stable Marriage: Second Optimization

```
% guess matching
{match(M,W) : woman(W)} = 1  :- man(M).

% no singles
married(M) :- match(M,W).
 :- woman(M), not married(M).

% strong stability condition
 :- match(M,W1), match(M1,W), W1 <> W,
    pref(M,W,Smw), pref(M,W1,Smw1), Smw > Smw1,
    pref(W,M,Swm), pref(W,M1,Swm1), Swm >= Swm1.
```

# Stable Marriage: Third Optimization

```
% guess matching
{match(M,W) : woman(W)} = 1  :-  man(M).

% no singles
married(M) :- match(M,W).
 :-  woman(M), not married(M).

% strong stability condition
  matched(m,M,S)        :- match(M,W), pref(M,W,S).
  matched(w,W,S-1)      :- match(M,W), pref(W,M,S), S > 1.
  matched(T,P,S-1)      :- matched(T,P,S), S > 1.
 :-  pref(M,W,R), pref(W,M,S), not matched(m,M,R), not
matched(w,W,S).
```

# Stable Marriage: Impact

- Can an efficient encoding make a huge difference in performance?

# Stable Marriage: Impact

- Can an efficient encoding make a huge difference in performance?
- Does an efficient encoding impact on performance or on number of rules?

# Stable Marriage: Impact

- Can an efficient encoding make a huge difference in performance?
- Does an efficient encoding impact on performance or on number of rules?
- Is this improvement limited to only one grounder?

# Stable Marriage: Impact

- Can an efficient encoding make a huge difference in performance?
- Does an efficient encoding impact on performance or on number of rules?
- Is this improvement limited to only one grounder?

**In practice (Tested on one instance from 3rd ASP Competition)**

# Stable Marriage: Impact

- Can an efficient encoding make a huge difference in performance?
- Does an efficient encoding impact on performance or on number of rules?
- Is this improvement limited to only one grounder?

**In practice (Tested on one instance from 3rd ASP Competition)**

| **Encoding** | **Time** | **Number of rules** |
|---|---|---|
| Natural encoding | 25 seconds | approx. 6 millions |

# Stable Marriage: Impact

- Can an efficient encoding make a huge difference in performance?
- Does an efficient encoding impact on performance or on number of rules?
- Is this improvement limited to only one grounder?

**In practice (Tested on one instance from 3rd ASP Competition)**

| **Encoding** | **Time** | **Number of rules** |
|---|---|---|
| Natural encoding | 25 seconds | approx. 6 millions |
| First optimization | 25 seconds | approx. 6 millions |

# Stable Marriage: Impact

- Can an efficient encoding make a huge difference in performance?
- Does an efficient encoding impact on performance or on number of rules?
- Is this improvement limited to only one grounder?

**In practice (Tested on one instance from 3rd ASP Competition)**

| **Encoding** | **Time** | **Number of rules** |
|---|---|---|
| Natural encoding | 25 seconds | approx. 6 millions |
| First optimization | 25 seconds | approx. 6 millions |
| Second optimization | 22 seconds | approx. 6 millions |

# Stable Marriage: Impact

- Can an efficient encoding make a huge difference in performance?
- Does an efficient encoding impact on performance or on number of rules?
- Is this improvement limited to only one grounder?

**In practice (Tested on one instance from 3rd ASP Competition)**

| **Encoding** | **Time** | **Number of rules** |
|---|---|---|
| Natural encoding | 25 seconds | approx. 6 millions |
| First optimization | 25 seconds | approx. 6 millions |
| Second optimization | 22 seconds | approx. 6 millions |
| Third optimization | 0.3 seconds | approx. 40 thousands |