

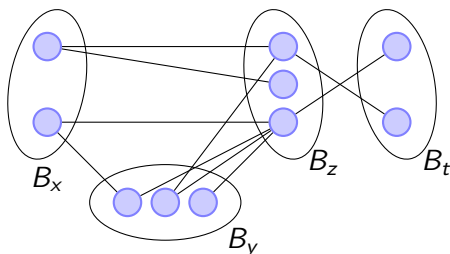
# Linear Datalog and $k$ -permutability = symmetric Datalog

Alexandr Kazda

Department of Mathematics  
Vanderbilt University  
Music City

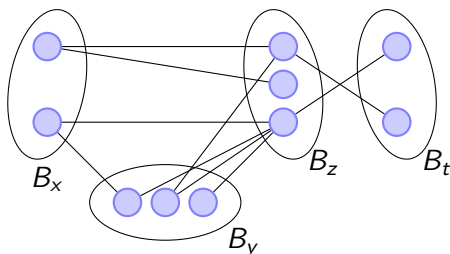
July 1, 2014

# Constraint Satisfaction Problem



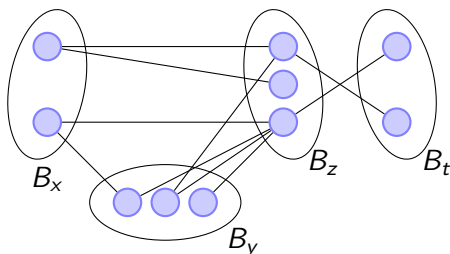
- An instance of  $\text{CSP}(\mathbb{A})$ . Unary and binary relations must be from the relational clone of  $\mathbb{A}$ .
- Is there a solution?

# Constraint Satisfaction Problem



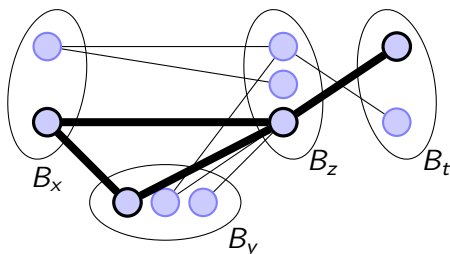
- An instance of  $\text{CSP}(\mathbb{A})$ . Unary and binary relations must be from the relational clone of  $\mathbb{A}$ .
- Is there a solution?

# Constraint Satisfaction Problem



- An instance of  $\text{CSP}(\mathbb{A})$ . Unary and binary relations must be from the relational clone of  $\mathbb{A}$ .
- Is there a solution?

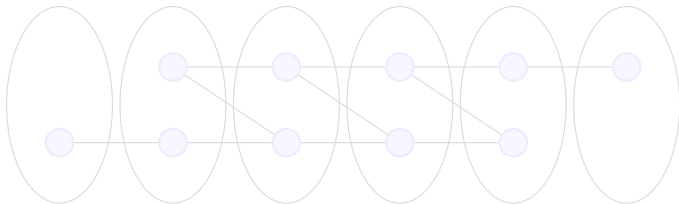
# Constraint Satisfaction Problem



- An instance of  $\text{CSP}(\mathbb{A})$ . Unary and binary relations must be from the relational clone of  $\mathbb{A}$ .
- Is there a solution?

# Consistency checking

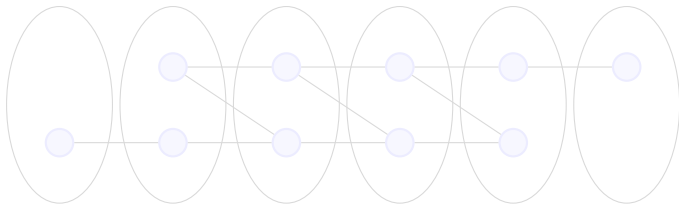
- Solving CSP is NP-complete in general.
- If a part of a CSP instance has no solution, then the whole instance has no solution.



- Good news: Consistency can be checked quickly.
- Bad news: An instance can have no solution, yet be locally consistent.

# Consistency checking

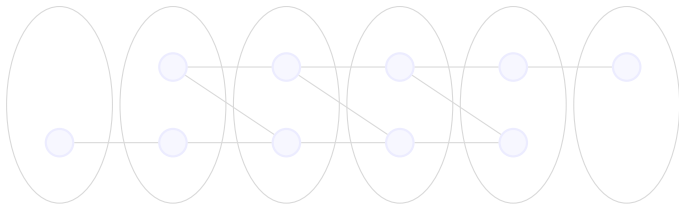
- Solving CSP is NP-complete in general.
- If a part of a CSP instance has no solution, then the whole instance has no solution.



- Good news: Consistency can be checked quickly.
- Bad news: An instance can have no solution, yet be locally consistent.

# Consistency checking

- Solving CSP is NP-complete in general.
- If a part of a CSP instance has no solution, then the whole instance has no solution.

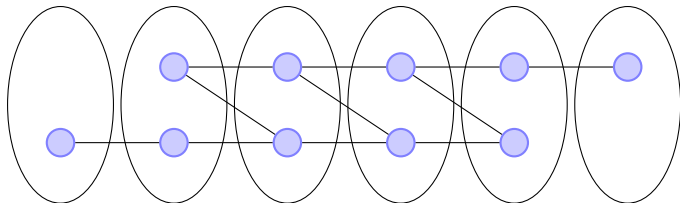


- Good news: Consistency can be checked quickly.
- Bad news: An instance can have no solution, yet be locally consistent.



# Consistency checking

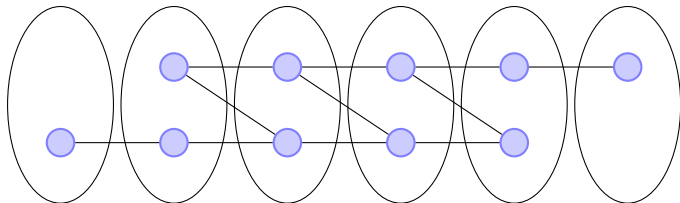
- Solving CSP is NP-complete in general.
- If a part of a CSP instance has no solution, then the whole instance has no solution.



- Good news: Consistency can be checked quickly.
- Bad news: An instance can have no solution, yet be locally consistent.

# Consistency checking

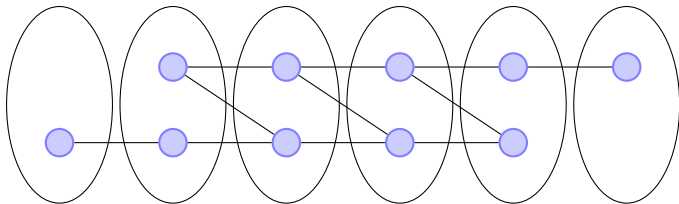
- Solving CSP is NP-complete in general.
- If a part of a CSP instance has no solution, then the whole instance has no solution.



- Good news: Consistency can be checked quickly.
- Bad news: An instance can have no solution, yet be locally consistent.

# Consistency checking

- Solving CSP is NP-complete in general.
- If a part of a CSP instance has no solution, then the whole instance has no solution.



- Good news: Consistency can be checked quickly.
- Bad news: An instance can have no solution, yet be locally consistent.

# Datalog

- A Datalog program adds tuples to predicates using local rules until it reaches its goal or can't apply any rule.
- Example of a Datalog program with unary predicates  $Z$ ,  $c_0$ ,  $c_1$ , binary predicate  $E$ , and the goal predicate  $G$ .

$$Z(x) \Leftarrow c_0(x)$$

$$Z(y) \Leftarrow Z(x) \wedge E(x, y)$$

$$G \Leftarrow Z(y) \wedge c_1(x)$$

- The above program checks if there is a directed path from something in  $c_0$  to something in  $c_1$ .
- How to use Datalog to solve  $\neg$ CSP: A Datalog program verifies local consistency. Goal: Prove inconsistency.

# Datalog

- A Datalog program adds tuples to predicates using local rules until it reaches its goal or can't apply any rule.
- Example of a Datalog program with unary predicates  $Z$ ,  $c_0$ ,  $c_1$ , binary predicate  $E$ , and the goal predicate  $G$ .

$$Z(x) \Leftarrow c_0(x)$$

$$Z(y) \Leftarrow Z(x) \wedge E(x, y)$$

$$G \Leftarrow Z(y) \wedge c_1(x)$$

- The above program checks if there is a directed path from something in  $c_0$  to something in  $c_1$ .
- How to use Datalog to solve  $\neg$ CSP: A Datalog program verifies local consistency. Goal: Prove inconsistency.

# Datalog

- A Datalog program adds tuples to predicates using local rules until it reaches its goal or can't apply any rule.
- Example of a Datalog program with unary predicates  $Z$ ,  $c_0$ ,  $c_1$ , binary predicate  $E$ , and the goal predicate  $G$ .

$$Z(x) \Leftarrow c_0(x)$$

$$Z(y) \Leftarrow Z(x) \wedge E(x, y)$$

$$G \Leftarrow Z(y) \wedge c_1(x)$$

- The above program checks if there is a directed path from something in  $c_0$  to something in  $c_1$ .
- How to use Datalog to solve  $\neg$ CSP: A Datalog program verifies local consistency. Goal: Prove inconsistency.

# Datalog

- A Datalog program adds tuples to predicates using local rules until it reaches its goal or can't apply any rule.
- Example of a Datalog program with unary predicates  $Z$ ,  $c_0$ ,  $c_1$ , binary predicate  $E$ , and the goal predicate  $G$ .

$$Z(x) \Leftarrow c_0(x)$$

$$Z(y) \Leftarrow Z(x) \wedge E(x, y)$$

$$G \Leftarrow Z(y) \wedge c_1(x)$$

- The above program checks if there is a directed path from something in  $c_0$  to something in  $c_1$ .
- How to use Datalog to solve  $\neg$ CSP: A Datalog program verifies local consistency. Goal: Prove inconsistency.

# Datalog

- A Datalog program adds tuples to predicates using local rules until it reaches its goal or can't apply any rule.
- Example of a Datalog program with unary predicates  $Z$ ,  $c_0$ ,  $c_1$ , binary predicate  $E$ , and the goal predicate  $G$ .

$$Z(x) \Leftarrow c_0(x)$$

$$Z(y) \Leftarrow Z(x) \wedge E(x, y)$$

$$G \Leftarrow Z(y) \wedge c_1(x)$$

- The above program checks if there is a directed path from something in  $c_0$  to something in  $c_1$ .
- How to use Datalog to solve  $\neg$ CSP: A Datalog program verifies local consistency. Goal: Prove inconsistency.



# Linear and Symmetric Datalog

- Predicates on left hand side of some rule: IDBs
- Linear Datalog: At most one IDB on the right hand side of any rule.
- Symmetric Datalog: At most one IDB on the right hand side of any rule. If there is an IDB on both sides of a rule, we can switch the IDBs.
- Example: Rule

$$Z(y) \leftarrow T(x) \wedge E(x, y)$$

gives

$$T(x) \leftarrow Z(y) \wedge E(x, y).$$

# Linear and Symmetric Datalog

- Predicates on left hand side of some rule: IDBs
- Linear Datalog: At most one IDB on the right hand side of any rule.
- Symmetric Datalog: At most one IDB on the right hand side of any rule. If there is an IDB on both sides of a rule, we can switch the IDBs.
- Example: Rule

$$Z(y) \leftarrow T(x) \wedge E(x, y)$$

gives

$$T(x) \leftarrow Z(y) \wedge E(x, y).$$

# Linear and Symmetric Datalog

- Predicates on left hand side of some rule: IDBs
- Linear Datalog: At most one IDB on the right hand side of any rule.
- Symmetric Datalog: At most one IDB on the right hand side of any rule. If there is an IDB on both sides of a rule, we can switch the IDBs.
- Example: Rule

$$Z(y) \leftarrow T(x) \wedge E(x, y)$$

gives

$$T(x) \leftarrow Z(y) \wedge E(x, y).$$

# Linear and Symmetric Datalog

- Predicates on left hand side of some rule: IDBs
- Linear Datalog: At most one IDB on the right hand side of any rule.
- Symmetric Datalog: At most one IDB on the right hand side of any rule. If there is an IDB on both sides of a rule, we can switch the IDBs.
- Example: Rule

$$Z(y) \Leftarrow T(x) \wedge E(x, y)$$

gives

$$T(x) \Leftarrow Z(y) \wedge E(x, y).$$

# Linear and Symmetric Datalog

- Predicates on left hand side of some rule: IDBs
- Linear Datalog: At most one IDB on the right hand side of any rule.
- Symmetric Datalog: At most one IDB on the right hand side of any rule. If there is an IDB on both sides of a rule, we can switch the IDBs.
- Example: Rule

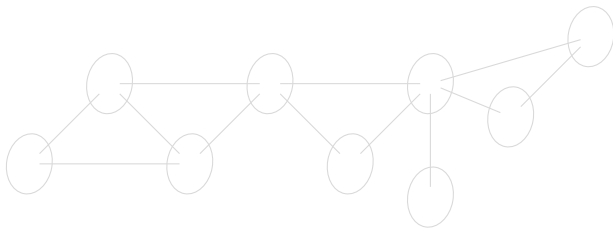
$$Z(y) \Leftarrow T(x) \wedge E(x, y)$$

gives

$$T(x) \Leftarrow Z(y) \wedge E(x, y).$$

# Bounded pathwidth duality

- There is a linear Datalog program solving  $\neg \text{CSP}(\mathbb{A})$  iff  $\mathbb{A}$  has bounded pathwidth duality.
- Bounded pathwidth duality: Unsatisfiable instances of  $\text{CSP}(\mathbb{A})$  always have unsatisfiable “path-like” parts.
- Example of what “path-like” means:

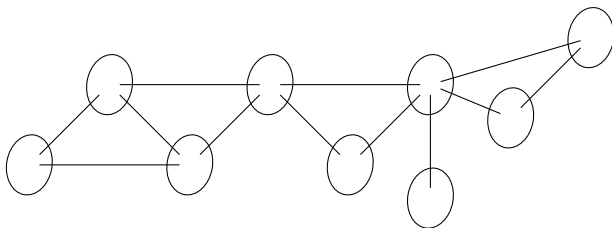


# Bounded pathwidth duality

- There is a linear Datalog program solving  $\neg \text{CSP}(\mathbb{A})$  iff  $\mathbb{A}$  has bounded pathwidth duality.
- Bounded pathwidth duality: Unsatisfiable instances of  $\text{CSP}(\mathbb{A})$  always have unsatisfiable “path-like” parts.
- Example of what “path-like” means:

# Bounded pathwidth duality

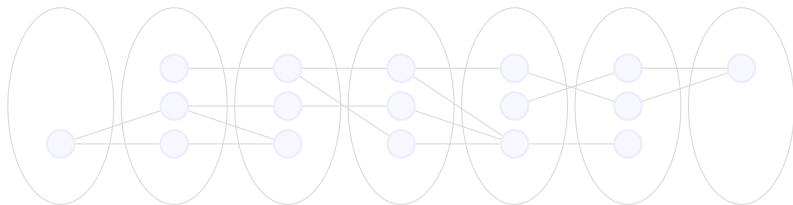
- There is a linear Datalog program solving  $\neg \text{CSP}(\mathbb{A})$  iff  $\mathbb{A}$  has bounded pathwidth duality.
- Bounded pathwidth duality: Unsatisfiable instances of  $\text{CSP}(\mathbb{A})$  always have unsatisfiable “path-like” parts.
- Example of what “path-like” means:





# Solving path instances is enough

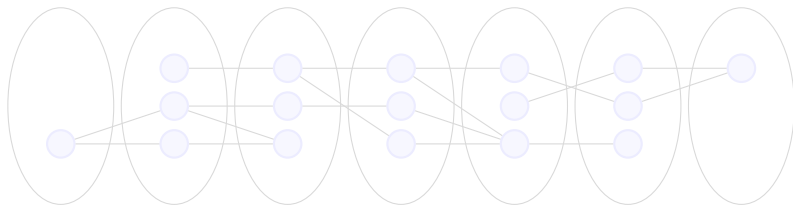
- Our goal: If  $\neg \text{CSP}(\mathbb{A})$  is solvable by a linear Datalog program and  $\mathbb{A}$  is  $n$ -permutable for some  $n$ , then  $\neg \text{CSP}(\mathbb{A})$  can be solved by a symmetric Datalog program.
- $\mathbb{A}$  has bounded pathwidth duality, so we look for path-like unsatisfiable instances.
- By replacing  $\mathbb{A}$  by its power, it is enough to look at path shaped unsatisfiable CSP instances.



- Our new goal: How to use symmetric Datalog to decide path CSP instances.

# Solving path instances is enough

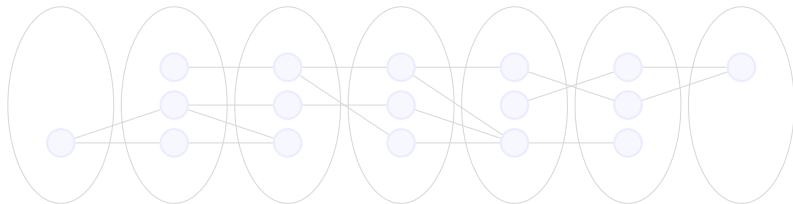
- Our goal: If  $\neg \text{CSP}(\mathbb{A})$  is solvable by a linear Datalog program and  $\mathbb{A}$  is  $n$ -permutable for some  $n$ , then  $\neg \text{CSP}(\mathbb{A})$  can be solved by a symmetric Datalog program.
- $\mathbb{A}$  has bounded pathwidth duality, so we look for path-like unsatisfiable instances.
- By replacing  $\mathbb{A}$  by its power, it is enough to look at path shaped unsatisfiable CSP instances.



- Our new goal: How to use symmetric Datalog to decide path CSP instances.

# Solving path instances is enough

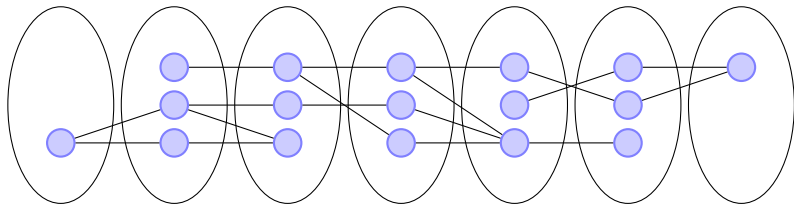
- Our goal: If  $\neg \text{CSP}(\mathbb{A})$  is solvable by a linear Datalog program and  $\mathbb{A}$  is  $n$ -permutable for some  $n$ , then  $\neg \text{CSP}(\mathbb{A})$  can be solved by a symmetric Datalog program.
- $\mathbb{A}$  has bounded pathwidth duality, so we look for path-like unsatisfiable instances.
- By replacing  $\mathbb{A}$  by its power, it is enough to look at path shaped unsatisfiable CSP instances.



- Our new goal: How to use symmetric Datalog to decide path CSP instances.

# Solving path instances is enough

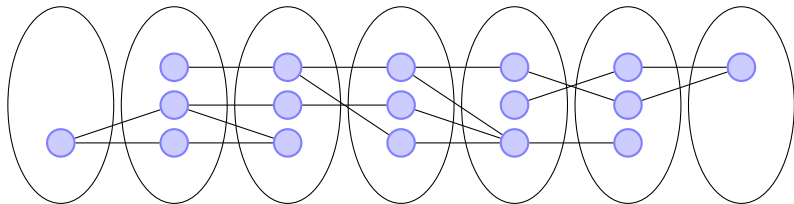
- Our goal: If  $\neg \text{CSP}(\mathbb{A})$  is solvable by a linear Datalog program and  $\mathbb{A}$  is  $n$ -permutable for some  $n$ , then  $\neg \text{CSP}(\mathbb{A})$  can be solved by a symmetric Datalog program.
- $\mathbb{A}$  has bounded pathwidth duality, so we look for path-like unsatisfiable instances.
- By replacing  $\mathbb{A}$  by its power, it is enough to look at path shaped unsatisfiable CSP instances.



- Our new goal: How to use symmetric Datalog to decide path CSP instances.

# Solving path instances is enough

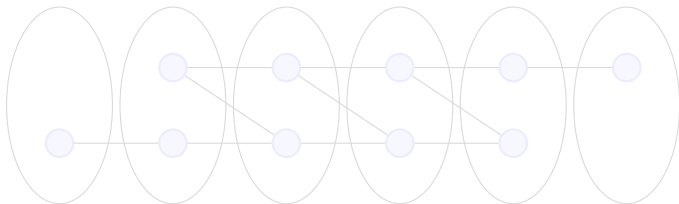
- Our goal: If  $\neg \text{CSP}(\mathbb{A})$  is solvable by a linear Datalog program and  $\mathbb{A}$  is  $n$ -permutable for some  $n$ , then  $\neg \text{CSP}(\mathbb{A})$  can be solved by a symmetric Datalog program.
- $\mathbb{A}$  has bounded pathwidth duality, so we look for path-like unsatisfiable instances.
- By replacing  $\mathbb{A}$  by its power, it is enough to look at path shaped unsatisfiable CSP instances.



- Our new goal: How to use symmetric Datalog to decide path CSP instances.

# Using $n$ -permutability

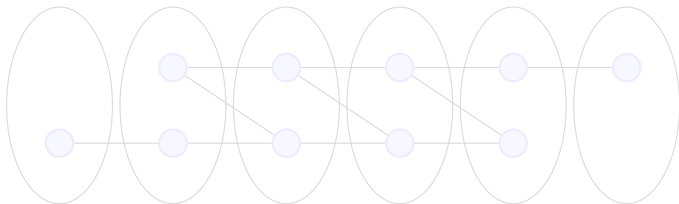
- If  $\mathbb{A}$  is 4-permutable, then the following can't be an unsatisfiable instance of  $\text{CSP}(\mathbb{A})$ :



- Applying Hagemann-Mitschke terms gives us a solution.

# Using $n$ -permutability

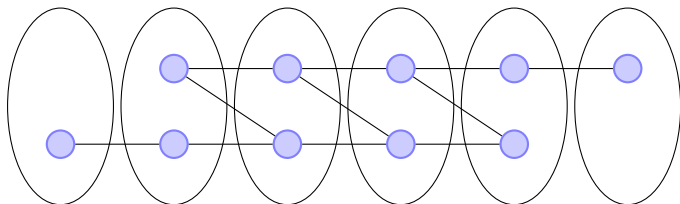
- If  $\mathbb{A}$  is 4-permutable, then the following can't be an unsatisfiable instance of  $\text{CSP}(\mathbb{A})$ :



- Applying Hagemann-Mitschke terms gives us a solution.

# Using $n$ -permutability

- If  $\mathbb{A}$  is 4-permutable, then the following can't be an unsatisfiable instance of  $\text{CSP}(\mathbb{A})$ :

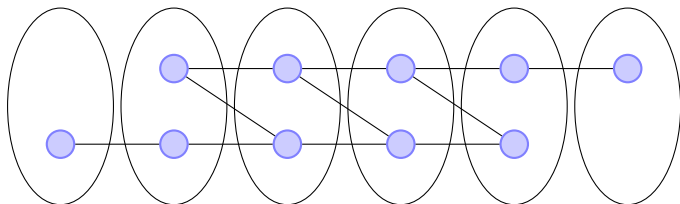


- Applying Hagemann-Mitschke terms gives us a solution.



# Using $n$ -permutability

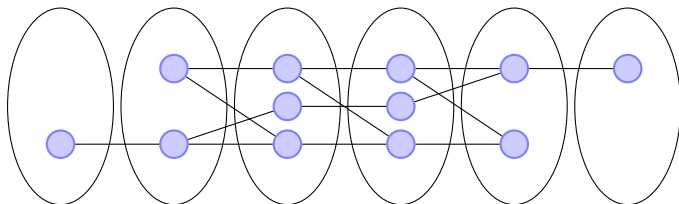
- If  $\mathbb{A}$  is 4-permutable, then the following can't be an unsatisfiable instance of  $\text{CSP}(\mathbb{A})$ :



- Applying Hagemann-Mitschke terms gives us a solution.

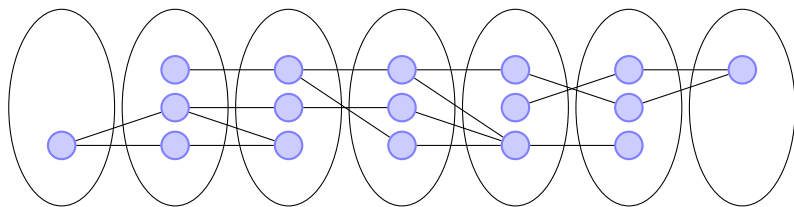
# Using $n$ -permutability

- If  $\mathbb{A}$  is 4-permutable, then the following can't be an unsatisfiable instance of  $\text{CSP}(\mathbb{A})$ :



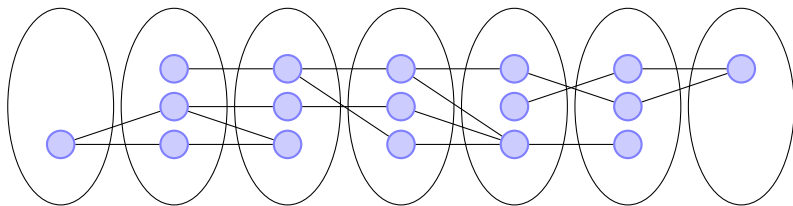
- Applying Hagemann-Mitschke terms gives us a solution.

# Glued potatoes



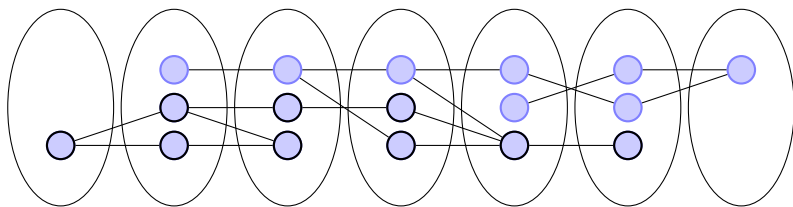
- We label black the vertices that can be reached from the starting potato.
- We want lots of backwards edges: Edges from blue to black vertices.
- We use some trickery to glue together potatoes without blue to black edges.
- Nature of the trickery: We can run a smaller symmetric Datalog program inside our original program.

# Glued potatoes



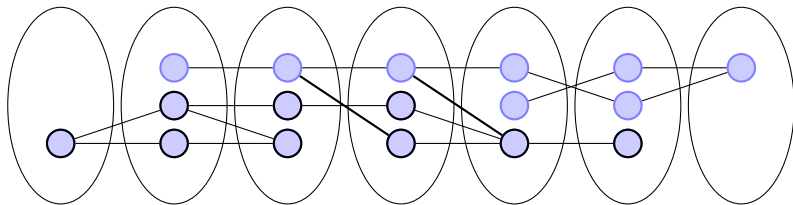
- We label black the vertices that can be reached from the starting potato.
- We want lots of backwards edges: Edges from blue to black vertices.
- We use some trickery to glue together potatoes without blue to black edges.
- Nature of the trickery: We can run a smaller symmetric Datalog program inside our original program.

# Glued potatoes



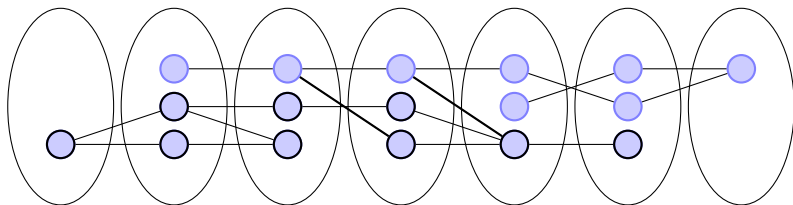
- We label black the vertices that can be reached from the starting potato.
- We want lots of backwards edges: Edges from blue to black vertices.
- We use some trickery to glue together potatoes without blue to black edges.
- Nature of the trickery: We can run a smaller symmetric Datalog program inside our original program.

# Glued potatoes



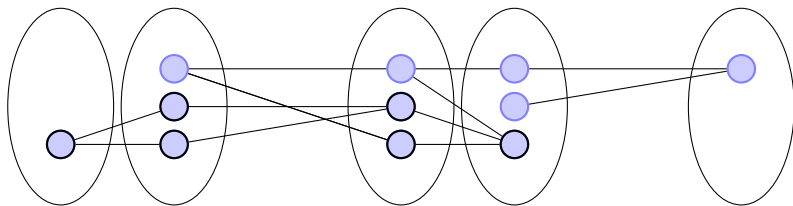
- We label black the vertices that can be reached from the starting potato.
- We want lots of backwards edges: Edges from blue to black vertices.
- We use some trickery to glue together potatoes without blue to black edges.
- Nature of the trickery: We can run a smaller symmetric Datalog program inside our original program.

# Glued potatoes



- We label black the vertices that can be reached from the starting potato.
- We want lots of backwards edges: Edges from blue to black vertices.
- We use some trickery to glue together potatoes without blue to black edges.
- Nature of the trickery: We can run a smaller symmetric Datalog program inside our original program.

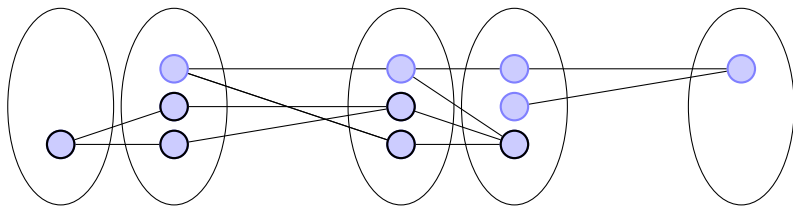
# Glued potatoes



- We label black the vertices that can be reached from the starting potato.
- We want lots of backwards edges: Edges from blue to black vertices.
- We use some trickery to glue together potatoes without blue to black edges.
- Nature of the trickery: We can run a smaller symmetric Datalog program inside our original program.



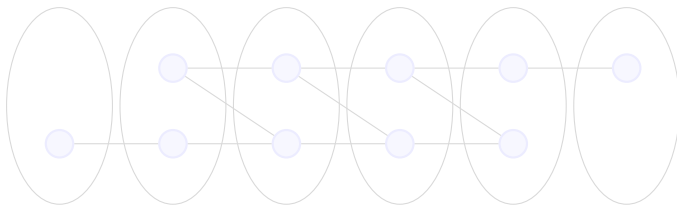
# Glued potatoes



- We label black the vertices that can be reached from the starting potato.
- We want lots of backwards edges: Edges from blue to black vertices.
- We use some trickery to glue together potatoes without blue to black edges.
- Nature of the trickery: We can run a smaller symmetric Datalog program inside our original program.

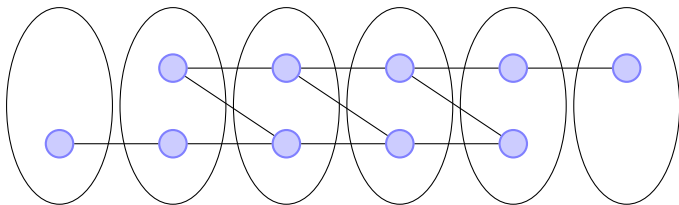
# Subdirect subinstance

- Additional trickery gives us a long part of the path instance where everything is subdirect.
- Then it is a matter of pigeonhole principle to find something like this:



# Subdirect subinstance

- Additional trickery gives us a long part of the path instance where everything is subdirect.
- Then it is a matter of pigeonhole principle to find something like this:



## Theorem

*Given  $n$ -permutatable relational structure  $\mathbb{A}$  such that  $\text{CSP}(\mathbb{A})$  has bounded pathwidth duality, there is a symmetric Datalog program that decides  $\text{CSP}(\mathbb{A})$ .*

- V. Dalmau previously proved this for 2-permutable structures.
- $n$ -permutability is necessary by L. Egri, B. Larose, and P. Tesson.

## Theorem

*Given  $n$ -permutatable relational structure  $\mathbb{A}$  such that  $\text{CSP}(\mathbb{A})$  has bounded pathwidth duality, there is a symmetric Datalog program that decides  $\text{CSP}(\mathbb{A})$ .*

- V. Dalmau previously proved this for 2-permutable structures.
- $n$ -permutability is necessary by L. Egri, B. Larose, and P. Tesson.

## Theorem

*Given  $n$ -permutatable relational structure  $\mathbb{A}$  such that  $\text{CSP}(\mathbb{A})$  has bounded pathwidth duality, there is a symmetric Datalog program that decides  $\text{CSP}(\mathbb{A})$ .*

- V. Dalmau previously proved this for 2-permutable structures.
- $n$ -permutability is necessary by L. Egri, B. Larose, and P. Tesson.

# Summing up, part 2

Algebraic view (due to L. Egri, B. Larose, and P. Tesson):

- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by linear Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive.
- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by symmetric Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive and  $n$ -permutable.

## Conjecture

*Semidistributive = linear Datalog = NL.*

## Conjecture

*Semidistributive and  $n$ -permutable = symmetric Datalog = L.*

- Our result plus first conjecture gives the second conjecture.

# Summing up, part 2

Algebraic view (due to L. Egri, B. Larose, and P. Tesson):

- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by linear Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive.
- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by symmetric Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive and  $n$ -permutable.

## Conjecture

*Semidistributive = linear Datalog = NL.*

## Conjecture

*Semidistributive and  $n$ -permutable = symmetric Datalog = L.*

- Our result plus first conjecture gives the second conjecture.



# Summing up, part 2

Algebraic view (due to L. Egri, B. Larose, and P. Tesson):

- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by linear Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive.
- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by symmetric Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive and  $n$ -permutable.

## Conjecture

*Semidistributive = linear Datalog = NL.*

## Conjecture

*Semidistributive and  $n$ -permutable = symmetric Datalog = L.*

- Our result plus first conjecture gives the second conjecture.

# Summing up, part 2

Algebraic view (due to L. Egri, B. Larose, and P. Tesson):

- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by linear Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive.
- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by symmetric Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive and  $n$ -permutable.

## Conjecture

*Semidistributive = linear Datalog = NL.*

## Conjecture

*Semidistributive and  $n$ -permutable = symmetric Datalog = L.*

- Our result plus first conjecture gives the second conjecture.

# Summing up, part 2

Algebraic view (due to L. Egri, B. Larose, and P. Tesson):

- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by linear Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive.
- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by symmetric Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive and  $n$ -permutable.

## Conjecture

*Semidistributive = linear Datalog = NL.*

## Conjecture

*Semidistributive and  $n$ -permutable = symmetric Datalog = L.*

- Our result plus first conjecture gives the second conjecture.

# Summing up, part 2

Algebraic view (due to L. Egri, B. Larose, and P. Tesson):

- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by linear Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive.
- If  $\neg \text{CSP}(\mathbb{A})$  is solvable by symmetric Datalog, then the algebra of  $\mathbb{A}$  must be (join) semidistributive and  $n$ -permutable.

## Conjecture

*Semidistributive = linear Datalog = NL.*

## Conjecture

*Semidistributive and  $n$ -permutable = symmetric Datalog = L.*

- Our result plus first conjecture gives the second conjecture.

Thank you for your attention.