Graphs of bounded treewidth as a generalisation of (undirected) trees:

- Trees have treewidth 1
- Graphs of higher treewidth resemble trees with “thicker branches”
- It is (in theory) not hard to check if a graph has treewidth \( \leq k \) for some \( k \)
- It is (in theory) not hard to answer BCQs whose primal graph has a bounded treewidth

Practically feasible only for lower treewidths

However, bounded treewidth does not generalise the notion of hypergraph acyclicity (acyclic families of hypergraphs may have unbounded treewidth)

Is there a better notion of tree-likeness for hypergraphs?
Query Width

Idea of Chekuri and Rajamaran [1997]:

- Create tree structure similar to tree decomposition
- But consider bags of query atoms instead of bags of variables
- Two connectedness conditions:
  1. Bags that refer to a certain variable must be connected
  2. Bags that refer to a certain query atom must be connected

Query width: least number of atoms needed in bags of a query decomposition
Query Width

Idea of Chekuri and Rajamaran [1997]:

- Create tree structure similar to tree decomposition
- But consider bags of query atoms instead of bags of variables
- Two connectedness conditions:
  1. Bags that refer to a certain variable must be connected
  2. Bags that refer to a certain query atom must be connected

Query width: least number of atoms needed in bags of a query decomposition

**Theorem 8.1:** Given a query decomposition for a BCQ, the query answering problem can be decided in time polynomial in the query width.
Problems with Query Width

Theorem 8.2 (Gottlob et al. 1999): Deciding if a query has query width at most $k$ is NP-complete.

In particular, it is also hard to find a query decomposition

$\leadsto$ Query answering complexity drops from NP to P . . .

. . . but we need to solve another NP-hard problem first!
Gottlob, Leone, and Scarcello had another idea on defining tree-like hypergraphs:

**Intuition:**

- Combine key ideas of tree decomposition and query decomposition
- Start by looking at a tree decomposition
- But define the width based on query atoms:
  How many atoms do we need to cover all variables in a bag?

$\leadsto$ Generalised hypertree width

$\leadsto$ A technical condition is needed to get a simpler-to-check notion
**Definition 8.3:** Consider a hypergraph $G = \langle V, E \rangle$. A hypertree decomposition of $G$ is a tree structure $T$ where each node $n$ of $T$ is associated with a bag of variables $B_n \subseteq V$ and with a set of edges $G_n \subseteq E$, such that:

- $T$ with $B_n$ yields a tree decomposition of the primal graph of $G$.
- For each node $n$ of $T$:
  1. the vertices used in the edges $G_n$ are a superset of $B_n$,
  2. if a vertex $v$ occurs in an edge of $G_n$ and this vertex also occurs in $B_m$ for some node $m$ below $n$ in $T$, then $v \in B_n$.

The width to $T$ is the largest number of edges in a set $G_n$.

The hypertree width of $G$, $\text{hw}(G)$, is the least width of its hypertree decompositions.

((2) is the “special condition”: without it we get the generalised hypertree width)
Hypertree Width: Example

[Diagram of a hypertree structure with nodes labeled 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, A, B, C, D, E, F, G, H.]

Markus Krötzsch, 7th May 2019

Database Theory
Hypertree Width: Example

\[ 1 \quad 2 \quad 6 \quad 4 \quad 7 \quad 5 \quad 10 \quad 8 \quad 9 \quad 3 \]

A, F  1, 2, 3, 6
C, F  1, 3, 4, 6, 10
B, H  3, 4, 6, 9, 10
C, E  4, 6, 8, 9, 10
B, G  4, 5, 6, 7, 8, 10

Markus Krötzsch, 7th May 2019
Hypertree Width: Example

1, 2, 3, 6
1, 3, 4, 6, 10
3, 4, 6, 9, 10
4, 6, 8, 9, 10
4, 5, 6, 7, 8, 10
A, F
C, F
B, H
C, E
B, G

Markus Krötzsch, 7th May 2019
Hypertree Width: Example

1, 2, 3, 6
1, 3, 4, 6, 10
3, 4, 6, 9, 10
4, 6, 8, 9, 10
4, 5, 6, 7, 8, 10

A, F
C, F
B, H
C, E
B, G

Markus Krötzsch, 7th May 2019
Hypertree Width: Example
Hypertree Width: Example

**Diagram:**
- Hypertree structure with nodes labeled 1 to 10, and edges connecting them.
- Node colors and labels correspond to hypertree widths:
  - Node 1: A, F, 1, 2, 3, 6
  - Node 2: A, F
  - Node 3: C, F
  - Node 4: B, H
  - Node 5: C, E
  - Node 6: B, G
  - Node 7: 3, 4, 6, 9, 10
  - Node 8: 4, 6, 8, 9, 10
  - Node 9: 4, 5, 6, 7, 8, 10
  - Node 10:
Hypertree Width: Example
Hypertree Width: Example

Special condition violated $\Rightarrow$ no hypertree decomposition
$\Rightarrow$ But generalised hypertree decomposition of width 2
Hypertree Width: Example
Hypertree Width: Example

1 2
6 4
7
5
10
8
9
3
A
C
H
G
B
F
E
D
C,F
B,G,H
1,2,3,4,6,10
3,4,5,6,7,8,9,10
Hypertree Width: Example

Special condition satisfied $\leadsto$ hypertree decomposition of width 3
Observation 8.4: If \( \langle T, (B_n), (G_n) \rangle \) is a hypertree decomposition for a hypergraph \( \langle V, E \rangle \), then the union of all sets \( G_n \) might be a proper subset of \( E \).

Proof: Indeed, we only require that every bag \( B_n \) is “covered” by the edges in \( G_n \), not that every edge in \( E \) is actually used for this purpose. \( \square \)
Observation 8.4: If \( \langle T, (B_n), (G_n) \rangle \) is a hypertree decomposition for a hypergraph \( \langle V, E \rangle \), then the union of all sets \( G_n \) might be a proper subset of \( E \).

Proof: Indeed, we only require that every bag \( B_n \) is “covered” by the edges in \( G_n \), not that every edge in \( E \) is actually used for this purpose. □

Observation 8.5: If \( \langle T, (B_n), (G_n) \rangle \) is a hypertree decomposition for a hypergraph \( \langle V, E \rangle \), then, for every hyperedge \( e \in E \), there is a node \( n \) in \( T \) such that \( e \subseteq B_n \).
Observation 8.4: If \( \langle T, (B_n), (G_n) \rangle \) is a hypertree decomposition for a hypergraph \( \langle V, E \rangle \), then the union of all sets \( G_n \) might be a proper subset of \( E \).

Proof: Indeed, we only require that every bag \( B_n \) is “covered” by the edges in \( G_n \), not that every edge in \( E \) is actually used for this purpose. \( \Box \)

Observation 8.5: If \( \langle T, (B_n), (G_n) \rangle \) is a hypertree decomposition for a hypergraph \( \langle V, E \rangle \), then, for every hyperedge \( e \in E \), there is a node \( n \) in \( T \) such that \( e \subseteq B_n \).

Proof: Since \( T, (B_n) \) is a tree decomposition of the primal graph, and every edge \( e \in E \) gives rise to a \(|e|\)-clique in this graph, the variables of \( e \) must occur together in one bag of the tree decomposition. \( \Box \)
Complete Hypertree Decompositions

We can make sure that all atoms are in fact used in some set $G_n$ of the decomposition:

**Theorem 8.6:** If $\langle T, (B_n), (G_n) \rangle$ is a (generalised) hypertree decomposition for a hypergraph $\langle V, E \rangle$, then there is a (generalised) hypertree decomposition $\langle T', (B'_n), (G'_n) \rangle$ of the same width and of size $O(|T| + |E|)$ such that, for all $e \in E$, there is a node $n$ in $T'$ with $e \in G'_n$. 

Proof:

For every edge $e \in E$ that does not appear in $(G_n)$ yet:

• extend $T$ with a new node $m$ that is a child of an existing node $n$ with $e \subseteq G_n$ (this must exist as just observed)

• define $B_m = e$ and $G_m = \{e\}$

This establishes the claim for $e$ and preserves all conditions in the definition of (generalised) hypertree decomposition. □

Such hypertree decompositions are called complete.
Complete Hypertree Decompositions

We can make sure that all atoms are in fact used in some set $G_n$ of the decomposition:

**Theorem 8.6:** If $\langle T, (B_n), (G_n) \rangle$ is a (generalised) hypertree decomposition for a hypergraph $\langle V, E \rangle$, then there is a (generalised) hypertree decomposition $\langle T', (B'_n), (G'_n) \rangle$ of the same width and of size $O(|T| + |E|)$ such that, for all $e \in E$, there is a node $n$ in $T'$ with $e \in G'_n$.

**Proof:** For every edge $e \in E$ that does not appear in $(G_n)$ yet:

- extend $T$ with a new node $m$ that is a child of an existing node $n$ with $e \subseteq G_n$ (this must exist as just observed)
- define $B_m = e$ and $G_m = \{e\}$

This establishes the claim for $e$ and preserves all conditions in the definition of (generalised) hypertree decomposition.

Such hypertree decompositions are called complete.
Theorem 8.7: A hypergraph is acyclic if and only if it has hypertree width 1.

Proof:

(⇒) Recall that an acyclic hypergraph has a join tree:
• A tree structure \( T \)
• Where each node is associated with a single edge
• Such that, for any vertex \( v \), the nodes with edges that mention \( v \) are a subtree of \( T \)

This easily corresponds to a hypertree decomposition (using the same tree structure, singleton edge sets \( G_n = \{e\} \) and vertex bags \( B_n = e \) if \( n \) is associated with \( e \)).
Theorem 8.7: A hypergraph is acyclic if and only if it has hypertree width 1.

Proof: (⇒)
Theorem 8.7: A hypergraph is acyclic if and only if it has hypertree width 1.

Proof: \( \implies \) Recall that an acyclic hypergraph has a join tree:

- A tree structure \( T \)
- where each node is associated with a single edge
- such that, for any vertex \( v \), the nodes with edges that mention \( v \) are a subtree of \( T \)
Theorem 8.7: A hypergraph is acyclic if and only if it has hypertree width 1.

Proof: ($\Rightarrow$) Recall that an acyclic hypergraph has a join tree:

- A tree structure $T$
- where each node is associated with a single edge
- such that, for any vertex $v$, the nodes with edges that mention $v$ are a subtree of $T$

This easily corresponds to a hypertree decomposition (using the same tree structure, singleton edge sets $G_n = \{e\}$ and vertex bags $B_n = e$ if $n$ is associated with $e$)
Theorem 8.7: A hypergraph is acyclic if and only if it has hypertree width 1.

Proof: \((\Leftarrow)\)
Theorem 8.7: A hypergraph is acyclic if and only if it has hypertree width 1.

Proof: $(\Leftarrow)$ For a hypergraph $\langle V, E \rangle$, consider a hypertree decomposition $\langle T, (B_n), (G_n) \rangle$ of width 1 that is complete (w.l.o.g.).
**Theorem 8.7:** A hypergraph is acyclic if and only if it has hypertree width 1.

**Proof:** ($\Leftarrow$) For a hypergraph $\langle V, E \rangle$, consider a hypertree decomposition $\langle T, (B_n), (G_n) \rangle$ of width 1 that is complete (w.l.o.g.).

We modify the decomposition so that, for every edge $e \in E$, there is exactly one node $n_e$ in $T$ such that $G_{n_e} = \{e\}$ and $B_{n_e} = e$:

- Choose an arbitrary total order $\prec$ on the nodes of $T$
- For each $e \in E$:
  - Find the $\prec$-least node $n_e$ of $T$ with $G_{n_e} = \{e\}$ and $B_{n_e} = e$
    (exists since we have a complete decomposition of width 1)
  - For every node $n$ with $G_n = \{e\}$: re-attach all children of $n$ to $n_e$ and delete $n$
Acyclic Hypergraphs and Hypertree Width (2)

**Theorem 8.7:** A hypergraph is acyclic if and only if it has hypertree width 1.

**Proof:** ($\iff$) For a hypergraph $\langle V, E \rangle$, consider a hypertree decomposition $\langle T, (B_n), (G_n) \rangle$ of width 1 that is complete (w.l.o.g.).

We modify the decomposition so that, for every edge $e \in E$, there is exactly one node $n_e$ in $T$ such that $G_{n_e} = \{e\}$ and $B_{n_e} = e$:

- Choose an arbitrary total order $\prec$ on the nodes of $T$
- For each $e \in E$:
  - Find the $\prec$-least node $n_e$ of $T$ with $G_{n_e} = \{e\}$ and $B_{n_e} = e$
    (exists since we have a complete decomposition of width 1)
  - For every node $n$ with $G_n = \{e\}$: re-attach all children of $n$ to $n_e$ and delete $n$

The modified hypertree decomposition corresponds to a join tree:

- each node is associated with a single edge
- no edge is associated with more than one node
- the vertices satisfy the connectedness condition for join trees
  (since $T$ is a tree decomposition of the primal graph)

Hence the hypergraph has a join tree and is therefore acyclic. ☐

Markus Krötzsch, 7th May 2019
Theorem 8.8: For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).
**Theorem 8.8:** For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

**Proof:** Consider a BCQ $q$, a width-$k$ hypertree decomposition $\langle T, (B_n), (G_n) \rangle$ of (the hypergraph of) $q$, and a database instance $I$. We first construct a modified BCQ $q'$, hypertree decomposition $\langle T, (B_n), (G'_n) \rangle$ of $q'$, and a database instance $I'$, such that $I|_{q} = q'$ if and only if $I'|_{q'} = q'$ and $\bigcup G'_n = B_n$ for all nodes $n$ of $T$:

- For each node $n$ and atom $r(\vec{x}) \in G_n$:
  - Create a new relation $r'$ and let $\vec{y}$ be a list of all variables in $\vec{x} \cap B_n$.
  - Replace $r(\vec{x}) \in G_n$ by $r'(\vec{y}) \in G'_n$.
  - Define $r'I'$ as the projection of $rI$ to $\vec{y}$. 

The BCQ $q'$, hypertree decomposition $\langle T, (B_n), (G'_n) \rangle$, and database instance $I'$ are of size polynomial in the input.
Efficient Query Answering

**Theorem 8.8:** For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

**Proof:** Consider a BCQ $q$, a width-$k$ hypertree decomposition $\langle T, (B_n), (G_n) \rangle$ of (the hypergraph of) $q$, and a database instance $\mathcal{I}$.

We first construct a modified BCQ $q'$, hypertree decomposition $\langle T, (B_n), (G'_n) \rangle$ of $q'$, and a database instance $\mathcal{I}'$, such that $\mathcal{I} \models q$ iff $\mathcal{I}' \models q'$ and $\bigcup G'_n = B_n$ for all nodes $n$ of $T$. 
**Theorem 8.8:** For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

**Proof:** Consider a BCQ $q$, a width-$k$ hypertree decomposition $\langle T, (B_n), (G_n) \rangle$ of (the hypergraph of) $q$, and a database instance $I$.

We first construct a modified BCQ $q'$, hypertree decomposition $\langle T, (B_n), (G'_n) \rangle$ of $q'$, and a database instance $I'$, such that $I \models q$ iff $I' \models q'$ and $\bigcup G'_n = B_n$ for all nodes $n$ of $T$:

- For each node $n$ and atom $r(\vec{x}) \in G_n$
- create a new relation $r'$ and let $\vec{y}$ be a list of all variables in $\vec{x} \cap B_n$
- replace $r(\vec{x}) \in G_n$ by $r'(\vec{y}) \in G'_n$
- define $r'^I$ as the projection of $r^I$ to $\vec{y}$
Theorem 8.8: For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

Proof: Consider a BCQ $q$, a width-$k$ hypertree decomposition $\langle T, (B_n), (G_n) \rangle$ of (the hypergraph of) $q$, and a database instance $I$.

We first construct a modified BCQ $q'$, hypertree decomposition $\langle T, (B_n), (G'_n) \rangle$ of $q'$, and a database instance $I'$, such that $I \models q$ iff $I' \models q'$ and $\bigcup G'_n = B_n$ for all nodes $n$ of $T$:

- For each node $n$ and atom $r(\vec{x}) \in G_n$
- create a new relation $r'$ and let $\vec{y}$ be a list of all variables in $\vec{x} \cap B_n$
- replace $r(\vec{x}) \in G_n$ by $r'(\vec{y}) \in G'_n$
- define $r'^I'$ as the projection of $r^I$ to $\vec{y}$

BCQ $q'$, hypertree decomposition $\langle T, (B_n), (G'_n) \rangle$, and database instance $I'$ are of size polynomial in the input.
Theorem 8.8: For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

Proof: We claim that $I \models q$ iff $I' \models q'$. 

• Each atom in $q'$ is just a restriction of an atom in $q$,

• The corresponding relation in $I'$ is a projection of the corresponding relation in $I$. 

(⇒) Every match of $q$ on $I$ is also a match of $q'$ on $I'$ since

(⇐) Every match of $q'$ in $I'$ is also a match of $q$ in $I$ since

For every atom $r(\vec{x})$ of $q$, there is a node $n$ of $T$ with $\vec{x} \subseteq B_n$ (observed before)

• so $r(\vec{x})$ is an atom of $q'$ as well.
Theorem 8.8: For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

Proof: We claim that $\mathcal{I} \models q$ iff $\mathcal{I}' \models q'$.

($\Rightarrow$) Every match of $q$ on $\mathcal{I}$ is also a match of $q'$ on $\mathcal{I}'$ since

- each atom in $q'$ is just a restriction of an atom in $q$, and
- the corresponding relation in $\mathcal{I}'$ is a projection of the corresponding relation in $\mathcal{I}$
Theorem 8.8: For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

Proof: We claim that $I \models q$ iff $I' \models q'$.

($\Rightarrow$) Every match of $q$ on $I$ is also a match of $q'$ on $I'$ since

- each atom in $q'$ is just a restriction of an atom in $q$, and
- the corresponding relation in $I'$ is a projection of the corresponding relation in $I$

($\Leftarrow$) Every match of $q'$ in $I'$ is also a match of $q$ in $I$ since

- For every atom $r(\vec{x})$ of $q$, there is a node $n$ of $T$ with $\vec{x} \subseteq B_n$ (observed before)
- so $r(\vec{x})$ is an atom of $q'$ as well
Theorem 8.8: For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

Proof: We now construct an acyclic BCQ $\bar{q}$, database $\bar{I}$, and join tree $J$ of $\bar{q}$, such that $I' \models q'$ iff $\bar{I} \models \bar{q}$.
**Theorem 8.8:** For a BCQ of (generalised) hypertree width \( k \), query answering can be decided in polynomial time (actually in \( \text{LOGCFL} \)).

**Proof:** We now construct an acyclic BCQ \( \bar{q} \), database \( \bar{I} \), and join tree \( J \) of \( \bar{q} \), such that \( I' \models q' \iff \bar{I} \models \bar{q} \).

- The tree structure of \( J \) is the same as \( T \)
- For each node \( n \) of \( T \):
  - we define a corresponding atom \( r_n(\bar{x}) \) of \( \bar{q} \) with variables \( \bar{x} = B_n \),
  - let \( r_n(\bar{x}) \) be the atom at the node of \( J \) that corresponds to \( n \), and
  - define \( r_n^{\bar{I}} \) to be the natural join of the atoms in \( G'_n \) over \( I' \)
Theorem 8.8: For a BCQ of (generalised) hypertree width \( k \), query answering can be decided in polynomial time (actually in LOGCFL).

Proof: We now construct an acyclic BCQ \( \bar{q} \), database \( \bar{I} \), and join tree \( J \) of \( \bar{q} \), such that \( \bar{I}' \models q' \) iff \( \bar{I} \models \bar{q} \).

- The tree structure of \( J \) is the same as \( T \)
- For each node \( n \) of \( T \):
  - we define a corresponding atom \( r_n(\vec{x}) \) of \( \bar{q} \) with variables \( \vec{x} = B_n \),
  - let \( r_n(\vec{x}) \) be the atom at the node of \( J \) that corresponds to \( n \), and
  - define \( r_n^{\bar{I}} \) to be the natural join of the atoms in \( G'_n \) over \( \bar{I}' \)

Observations:
- The outcome is polynomial in size
- We find \( \bar{I}' \models q' \) iff \( \bar{I} \models \bar{q} \)
Theorem 8.8: For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time (actually in LOGCFL).

Proof: We now construct an acyclic BCQ $\bar{q}$, database $\bar{I}$, and join tree $J$ of $\bar{q}$, such that $I' \models q'$ iff $I \models \bar{q}$.

• The tree structure of $J$ is the same as $T$
• For each node $n$ of $T$:
  – we define a corresponding atom $r_n(\vec{x})$ of $\bar{q}$ with variables $\vec{x} = B_n$,
  – let $r_n(\vec{x})$ be the atom at the node of $J$ that corresponds to $n$, and
  – define $r_n^{\bar{I}}$ to be the natural join of the atoms in $G'_n$ over $I'$

Observations:
  – The outcome is polynomial in size
  – We find $I' \models q'$ iff $I \models \bar{q}$

The overall claim now follows by applying Yannakakis' Algorithm to answer the query. \qed
Hypertree Width: Results

• Relationships of hypergraph tree-likeness measures:
  generalised hypertree width $\leq$ hypertree width $\leq$ query width
  (both inequalities might be $<$ in some cases)
Hypertree Width: Results

• Relationships of hypergraph tree-likeness measures:
  generalised hypertree width ≤ hypertree width ≤ query width
  (both inequalities might be < in some cases)

• Acyclic graphs have hypertree width 1
Hypertree Width: Results

• Relationships of hypergraph tree-likeness measures:
  generalised hypertree width $\leq$ hypertree width $\leq$ query width
  (both inequalities might be $<$ in some cases)

• Acyclic graphs have hypertree width 1

• Deciding “query width $< k$?” is NP-complete
Hypertree Width: Results

• Relationships of hypergraph tree-likeness measures:
  generalised hypertree width $\leq$ hypertree width $\leq$ query width
  (both inequalities might be $<$ in some cases)

• Acyclic graphs have hypertree width 1

• Deciding “query width $< k$?” is NP-complete

• Deciding “generalised hypertree width $< 4$?” is NP-complete
Hypertree Width: Results

- Relationships of hypergraph tree-likeness measures:
  generalised hypertree width $\leq$ hypertree width $\leq$ query width
  (both inequalities might be $<$ in some cases)
- Acyclic graphs have hypertree width 1
- Deciding “query width $< k$?” is NP-complete
- Deciding “generalised hypertree width $< 4$?” is NP-complete
- Deciding “hypertree width $< k$?” is polynomial (LOGCFL)
Hypertree Width: Results

• Relationships of hypergraph tree-likeness measures:
  generalised hypertree width ≤ hypertree width ≤ query width
  (both inequalities might be < in some cases)
• Acyclic graphs have hypertree width 1
• Deciding “query width < k?” is NP-complete
• Deciding “generalised hypertree width < 4?” is NP-complete
• Deciding “hypertree width < k?” is polynomial (LOGCFL)
• Hypertree decompositions can be computed in polynomial time if k is fixed
Hypertree Width: Results

- Relationships of hypergraph tree-likeness measures:
  - generalised hypertree width $\leq$ hypertree width $\leq$ query width
  (both inequalities might be $<$ in some cases)
- Acyclic graphs have hypertree width 1
- Deciding “query width $< k$?” is NP-complete
- Deciding “generalised hypertree width $< 4$?” is NP-complete
- Deciding “hypertree width $< k$?” is polynomial (LOGCFL)
- Hypertree decompositions can be computed in polynomial time if $k$ is fixed

**Theorem 8.9:** For a BCQ of (generalised) hypertree width $k$, query answering can be decided in polynomial time, and is complete for LOGCFL.

... but the degree of the polynomial time bound is greater than $k$
Hypertree Width via Games

There is also a game characterisation of (generalised) hypertree width.

The Marshals-and-Robber Game

- The game is played on a hypergraph
- There are $k$ marshals, each controlling one hyperedge, and one robber located at a vertex
- Otherwise similar to cops-and-robber game
- Special condition: Marshals must shrink the space that is left for the robber in every turn!

Hypertree width $\leq k$ if and only if $k$ marshals have a winning strategy
$\rightsquigarrow$ hypergraph is acyclic iff 1 marshal has a winning strategy
Hypertree Width via Logic

There is also a logical characterisation of hypertree width.

Loosely $k$-Guarded Logic

- Fragment of FO with $\exists$ and $\land$
- Special form for all $\exists$ subexpressions:

$$
\exists x_1, \ldots, x_n. (G_1 \land \ldots \land G_k \land \varphi)
$$

where $G_i$ are atoms ("guards") and every variable that is free in $\varphi$ occurs in one such atom $G_i$.

A query has hypertree width $\leq k$ if and only if it can be expressed as a loosely $k$-guarded formula

$\sim$ tree queries correspond to loosely 1-guarded formulae

("loosely 1-guarded" logic is better known as guarded logic and widely studied)
Summary and Outlook

Besides tree queries, there are other important classes of CQs that can be answered in polynomial time:

- Bounded treewidth queries
- Bounded hypertree width queries

General idea: decompose the query in a tree structure

Other possible characterisations via games and logic

Open questions:

- What else is there besides query answering? → optimisation
- Measure expressivity rather than just complexity