A Practical Algorithm for Structure Embedding

Charlie Murphy

Overview

1. Structure Embedding

2. Use in Multi-threaded Verification

3. MatchEmbeds

Overview

1. Structure Embedding

2. Use in Multi-threaded Verification

3. MatchEmbeds

Structures

- Finite relational **structure** $\langle \mathcal{U}, \mathcal{R} \rangle$:

Structures

- Finite relational **structure** $\langle \mathcal{U}, \mathcal{R} \rangle$:
  - $\mathcal{U}$: finite universe of elements

Structures

- Finite relational **structure** $\langle \mathcal{U}, \mathcal{R} \rangle$:
  - $\mathcal{U}$: finite universe of elements
  - $\mathcal{R}$: finite set of relations over elements of $\mathcal{U}$

Structures

- Finite relational **structure** $\langle \mathcal{U}, \mathcal{R} \rangle$:
  - $\mathcal{U}$: finite universe of elements
  - $\mathcal{R}$: finite set of relations over elements of $\mathcal{U}$

- Examples:

Structures

- Finite relational **structure** $\langle \mathcal{U}, \mathcal{R} \rangle$:
  - $\mathcal{U}$: finite universe of elements
  - $\mathcal{R}$: finite set of relations over elements of $\mathcal{U}$

- Examples:
  - Graph $\equiv \langle V, edge \rangle$

Structures

- Finite relational **structure** $\langle \mathcal{U}, \mathcal{R} \rangle$:
  - $\mathcal{U}$: finite universe of elements
  - $\mathcal{R}$: finite set of relations over elements of $\mathcal{U}$

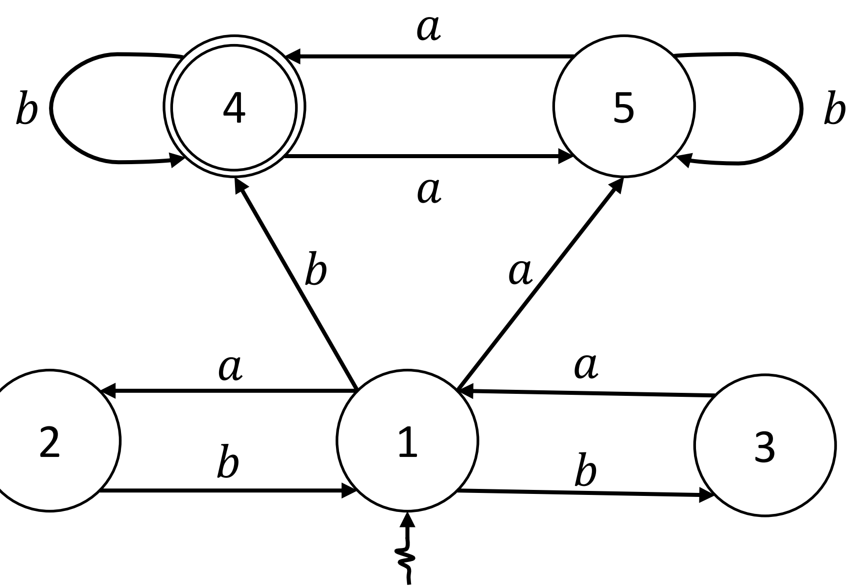
- Examples:
  - Graph $\equiv \langle V, edge \rangle$
  - NFA $\equiv \langle S, \{final, start\} \cup \{\Delta_a : a \in \Sigma\} \rangle$

Structures

- Finite relational **structure** $\langle \mathcal{U}, \mathcal{R} \rangle$:
  - $\mathcal{U}$: finite universe of elements
  - $\mathcal{R}$: finite set of relations over elements of $\mathcal{U}$

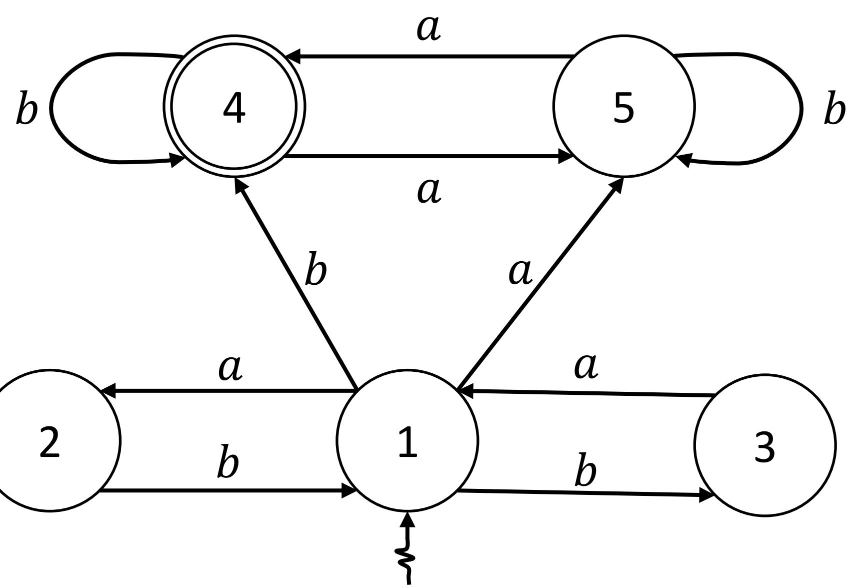
- Examples:
  - Graph $\equiv \langle V, edge \rangle$
  - NFA $\equiv \langle S, \{final, start\} \cup \{\Delta_a : a \in \Sigma\} \rangle$
  - Database $\equiv \langle Values, \{table_1, \dots, table_n\} \rangle$

Structures



$$\mathfrak{F} \stackrel{\text{def}}{=} \langle \{1,2,3,4,5\}, Start, Final, \Delta_a, \Delta_b \rangle$$

Structures



$$\mathfrak{F} \stackrel{\text{def}}{=} \langle \{1,2,3,4,5\}, Start, Final, \Delta_a, \Delta_b \rangle$$

where:

$Start \stackrel{\text{def}}{=} \{1\}$

$Final \stackrel{\text{def}}{=} \{4\}$

$\Delta_a \stackrel{\text{def}}{=} \{\langle 1,2 \rangle, \langle 1,5 \rangle, \langle 3,1 \rangle, \langle 4.5 \rangle, \langle 5,4 \rangle\}$

$\Delta_b \stackrel{\text{def}}{=} \{\langle 1,3 \rangle, \langle 1,4 \rangle, \langle 2,1 \rangle, \langle 4,4 \rangle, \langle 5,5 \rangle\}$

Structure Embedding

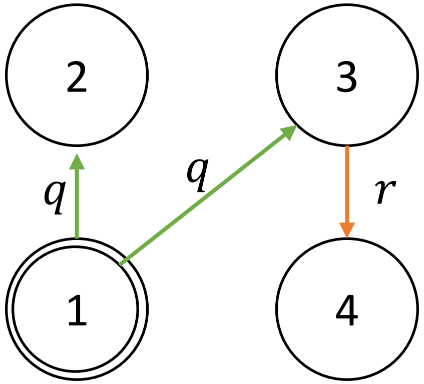
$$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3,4\}, p^{\mathfrak{A}}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$$

$$p^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$$

$$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{\langle 1,2\rangle, \langle 1,3\rangle\}$$

$$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{\langle 3,4\rangle\}$$

$\mathfrak{A}$:



Structure Embedding

$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3,4\}, p^{\mathfrak{A}}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$

$p^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{\langle 1,2\rangle, \langle 1,3\rangle\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{\langle 3,4\rangle\}$

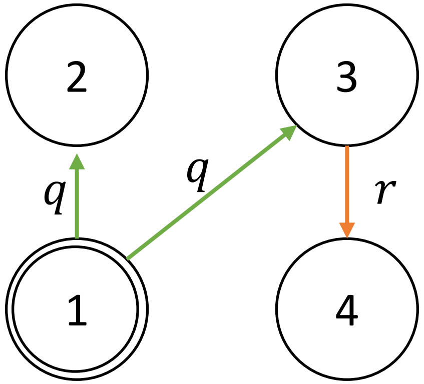
$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3,4\}, p^{\mathfrak{B}}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$

$p^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$

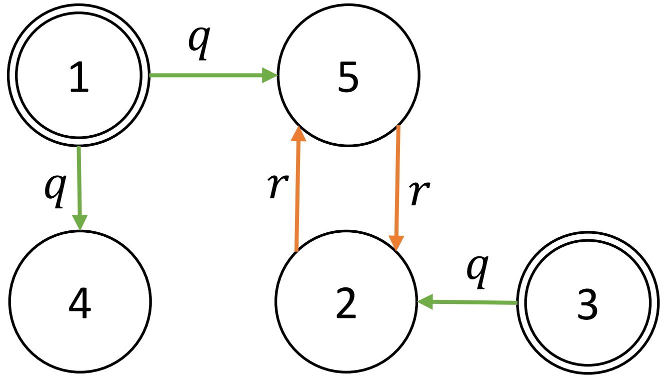
$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{\langle 1,4\rangle, \langle 1,5\rangle, \langle 3,2\rangle\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{\langle 2,5\rangle, \langle 5,2\rangle\}$

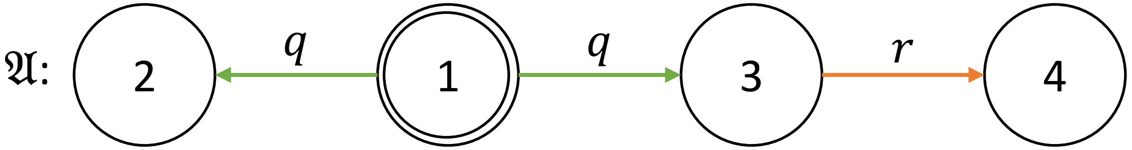
$\mathfrak{A}$:



$\mathfrak{B}$:



Structure Embedding

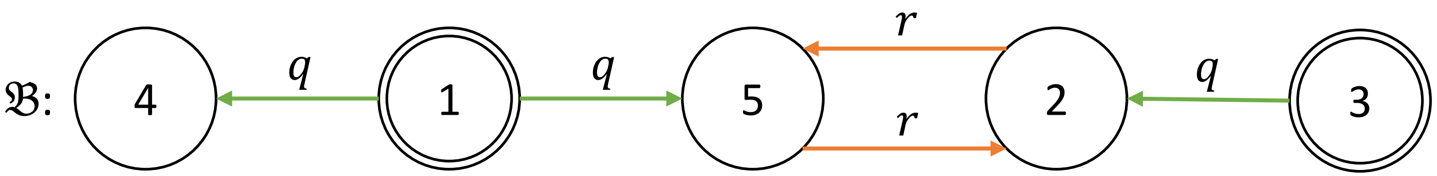


$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3,4\}, p^{\mathfrak{A}}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$

$p^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{\langle 1,2 \rangle, \langle 1,3 \rangle\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{\langle 3,4 \rangle\}$



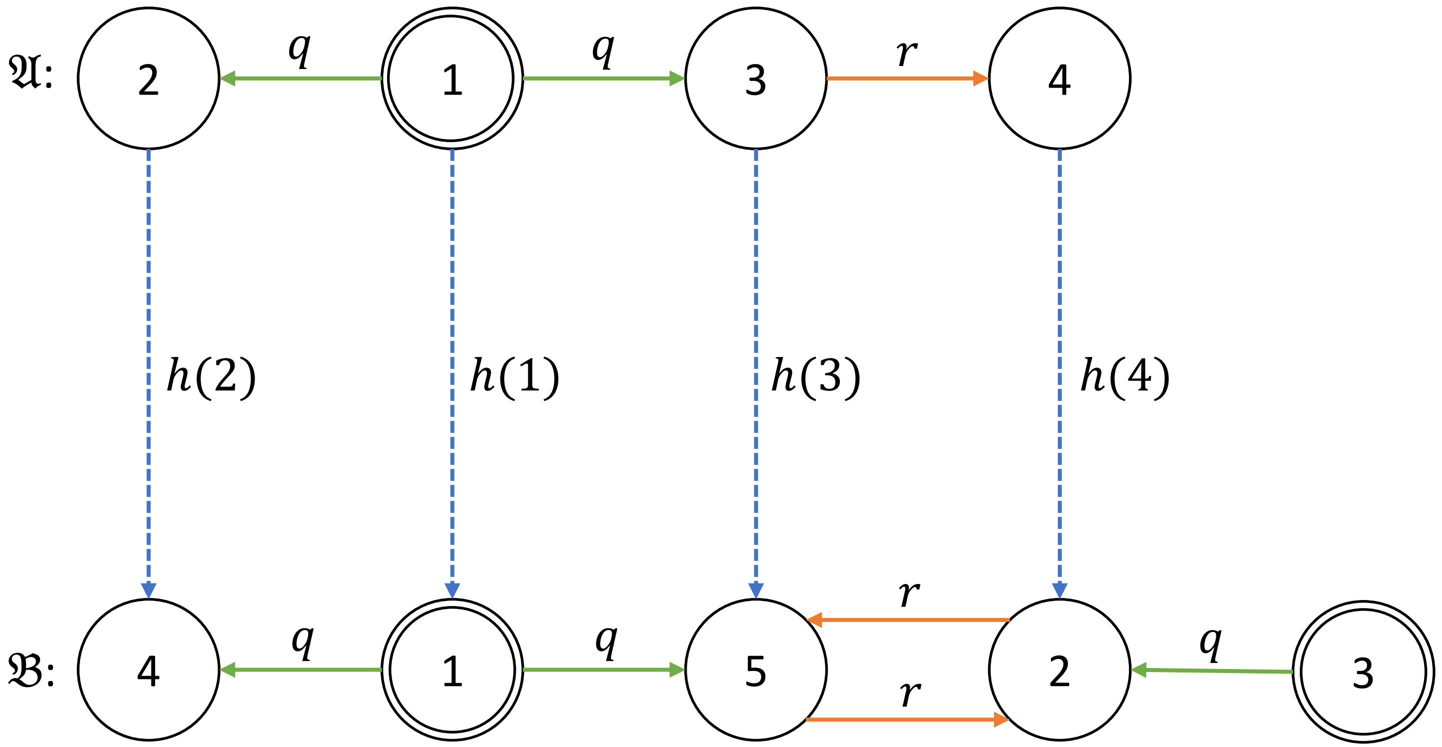
$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3,4\}, p^{\mathfrak{B}}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$

$p^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$

$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{\langle 1,4 \rangle, \langle 1,5 \rangle, \langle 3,2 \rangle\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{\langle 2,5 \rangle, \langle 5,2 \rangle\}$

Structure Embedding



$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3,4\}, p^{\mathfrak{A}}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$

$p^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{\langle 1,2\rangle, \langle 1,3\rangle\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{\langle 3,4\rangle\}$

$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3,4\}, p^{\mathfrak{B}}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$

$p^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$

$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{\langle 1,4\rangle, \langle 1,5\rangle, \langle 3,2\rangle\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{\langle 2,5\rangle, \langle 5,2\rangle\}$

Structure Embedding

- Given $\mathfrak{A}$ and $\mathfrak{B}$ over a common **vocabulary** $\langle Q, ar \rangle$

Structure Embedding

- Given $\mathfrak{A}$ and $\mathfrak{B}$ over a common **vocabulary** $\langle Q, ar \rangle$
  - A **homomorphism** is a function $h : A \to B$

Structure Embedding

- Given $\mathfrak{A}$ and $\mathfrak{B}$ over a common **vocabulary** $\langle Q, ar \rangle$
  - A **homomorphism** is a function $h : A \to B$
    - $\forall q \in Q. \langle a_1, \dots, a_{ar(q)} \rangle \in q^{\mathfrak{A}} \implies \langle h(a_1), \dots, h(a_{ar(q)}) \rangle \in q^{\mathfrak{B}}$

Structure Embedding

- Given $\mathfrak{A}$ and $\mathfrak{B}$ over a common **vocabulary** $\langle Q, ar \rangle$
  - A **homomorphism** is a function $h : A \to B$
    - $\forall q \in Q. \langle a_1, \ldots, a_{ar(q)} \rangle \in q^{\mathfrak{A}} \implies \langle h(a_1), \ldots, h(a_{ar(q)}) \rangle \in q^{\mathfrak{B}}$

  - An **embedding** is an injective homomorphism

Structure Embedding

- Given $\mathfrak{A}$ and $\mathfrak{B}$ over a common **vocabulary** $\langle Q, ar \rangle$
  - A **homomorphism** is a function $h : A \rightarrow B$
    - $\forall q \in Q. \langle a_1, \ldots, a_{ar(q)} \rangle \in q^{\mathfrak{A}} \implies \langle h(a_1), \ldots, h(a_{ar(q)}) \rangle \in q^{\mathfrak{B}}$
  - An **embedding** is an injective homomorphism
  - Structure Embedding Problem:
    - Given $\mathfrak{A}$ and $\mathfrak{B}$ determine if $\mathfrak{A}$ embeds into $\mathfrak{B}$

Structure Embedding

- Given $\mathfrak{A}$ and $\mathfrak{B}$ over a common **vocabulary** $\langle Q, ar\rangle$
  - A **homomorphism** is a function $h : A \rightarrow B$
    - $\forall q \in Q. \langle a_1, \ldots, a_{ar(q)}\rangle \in q^{\mathfrak{A}} \implies \langle h(a_1), \ldots, h(a_{ar(q)})\rangle \in q^{\mathfrak{B}}$
  - An **embedding** is an injective homomorphism
  - Structure Embedding Problem:
    - Given $\mathfrak{A}$ and $\mathfrak{B}$ determine if $\mathfrak{A}$ embeds into $\mathfrak{B}$
    - NP-Complete

Contributions

- MatchEmbeds
    - Structure Embedding Problem

Contributions

- MatchEmbeds
  - Structure Embedding Problem
    - NP Complete

Contributions

- MatchEmbeds
  - Structure Embedding Problem
    - NP Complete
    - Occurs during verification of multi-threaded programs
      - Many (1000’s) embedding queries are often required

Contributions

- MatchEmbeds
  - Structure Embedding Problem
    - NP Complete
    - Occurs during verification of multi-threaded programs
      - Many (1000’s) embedding queries are often required
      - Mostly monadic predicates
      - Most involve only a small number of threads

Contributions

- MatchEmbeds
  - Structure Embedding Problem
    - NP Complete
    - Occurs during verification of multi-threaded programs
      - Many (1000’s) embedding queries are often required
      - Mostly monadic predicates
      - Most involve only a small number of threads
  - Backtracking search

Contributions

- MatchEmbeds
  - Structure Embedding Problem
    - NP Complete
    - Occurs during verification of multi-threaded programs
      - Many (1000’s) embedding queries are often required
      - Mostly monadic predicates
      - Most involve only a small number of threads
  - Backtracking search
    - Polytime for monadic case

Contributions

- MatchEmbeds
  - Structure Embedding Problem
    - NP Complete
    - Occurs during verification of multi-threaded programs
      - Many (1000's) embedding queries are often required
      - Mostly monadic predicates
      - Most involve only a small number of threads
  - Backtracking search
    - Polytime for monadic case
    - Practical for "real life" instances
    - Solves difficult instances quickly

Overview

1. Structure Embedding

2. Use in Multi-threaded Verification

3. MatchEmbeds

Multi-threaded Program Verification

```
main_count():
  count = 0
  for i = 1 to N:
    fork thread_count
  assert(count ≤ N)

thread_count():
  count = count+1
```

Multi-threaded Program Verification

```
main_count():
  count = 0
  for i = 1 to N:
    fork thread_count
  assert(count≤N)

thread_count():
  count = count+1
```

```
main_ticket():
  s = t = 0
  while (*)
   fork thread_ticket

thread_ticket():
  local m
  m = t++
  while (s < m); skip
  //mutual exclusion
  s++
```

Predicate Abstraction

- Represent program states by conjunction of predicates

Predicate Abstraction

- Represent program states by conjunction of predicates

```
Fib(a, b, n):
 while (n > 0)
  tmp = a + b
  a = b
  b = tmp
  n--
 return a
```

Predicate Abstraction

- Represent program states by conjunction of predicates

```
  Fib(a, b, n):
1  while (n > 0)
2    tmp = a + b
3    a = b
4    b = tmp
5    n--
6  return a
```

Predicate Abstraction

$$(pc = 3) \land (n > 0) \land (tmp \geq 2a) \land (a < b)$$

Predicate Abstraction

- Represent program states by conjunction of predicates

```
  Fib(a, b, n):
1  while (n > 0)
2    tmp = a + b
3    a = b
4    b = tmp
5    n--
6  return a
```

Predicate Abstraction

$$(pc = 3) \land (n > 0) \land (tmp \geq 2a) \land (a < b)$$

What about multi-threaded programs?

Predicate Abstraction

- Represent program states by structures

Predicate Abstraction

- Represent program states by structures

```
main_ticket():
s = t = 0
while (*)
 fork thread_ticket

thread_ticket():
local m
m = t++
while (s < m); skip
//mutual exclusion
s++
```

Predicate Abstraction

- Represent program states by structures

```
main_ticket():
s = t = 0
while (*)
 fork thread_ticket

thread_ticket():
local m
m = t++
while (s < m); skip
//mutual exclusion
s++
```

Relational vocabulary $\langle Q, ar \rangle$

$Q = \{l_i, S_{lt}, M_{lt}, \}$

$ar(l_i) = ar(S_{lt}) = 1, ar(M_{lt})$

Predicate Abstraction

- Represent program states by structures

```
main_ticket():
1 s = t = 0
2 while (*)
3   fork thread_ticket

thread_ticket():
4 local m
5 m = t++
6 while (s < m); skip
7 //mutual exclusion
8 s++
```

Relational vocabulary $\langle Q, ar \rangle$

$Q = \{l_i, S_{lt}, M_{lt}, \}$

$ar(l_i) = ar(S_{lt}) = 1, ar(M_{lt}) = 2$

$l_4(j) \stackrel{\text{def}}{=}$ thread j is at location 4

$S_{lt}(j) \stackrel{\text{def}}{=} s < m_j$

$M_{lt}(i,j) \stackrel{\text{def}}{=} m_i < m_j$

Predicate Abstraction

- Represent program states by structures

```
 main_ticket():
1 s = t = 0
2 while (*)
3  fork thread_ticket

 thread_ticket():
4 local m
5 m = t++
6 while (s < m); skip
7 //mutual exclusion
8 s++
```

Relational vocabulary $\langle Q, ar \rangle$

$$Q = \{l_i, S_{lt}, M_{lt}, \}$$
$$ar(l_i) = ar(S_{lt}) = 1, ar(M_{lt}) = 2$$

$$l_4(1) \wedge l_6(2) \wedge l_7(3) \wedge S_{lt}(2) \wedge M_{lt}(2,3)$$

$l_4(j) \stackrel{\text{def}}{=}$ thread j is at location 4

$$S_{lt}(j) \stackrel{\text{def}}{=} s < m_j$$
$$M_{lt}(i,j) \stackrel{\text{def}}{=} m_i < m_j$$

Predicate Automata

- Automata used to verify safety of multi-threaded programs

Predicate Automata

- Automata used to verify safety of multi-threaded programs
  - Structures represent program state

Predicate Automata

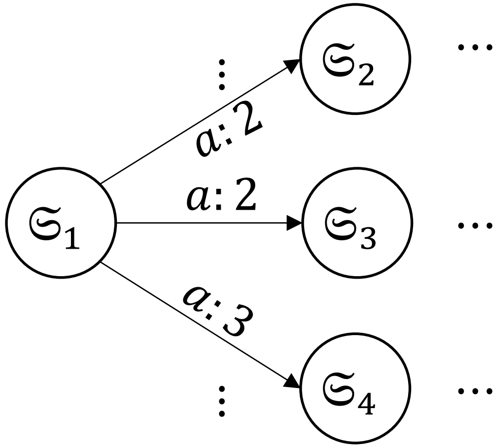
- Automata used to verify safety of multi-threaded programs
  - Structures represent program state
  - Program statements transition between structures

Predicate Automata

- Automata used to verify safety of multi-threaded programs
  - Structures represent program state
  - Program statements transition between structures
  - Program safety is reduced to checking emptiness of a PA

Predicate Automata

- Automata used to verify safety of multi-threaded programs
  - Structures represent program state
  - Program statements transition between structures
  - Program safety is reduced to checking emptiness of a PA
- Infinite state automata over infinite alphabet ($\Sigma \times \mathbb{N}$)



Emptiness Checking

- Determine if an accepting structure is reachable

Emptiness Checking

- Determine if an accepting structure is reachable
- Undecidable in general

Emptiness Checking

- Determine if an accepting structure is reachable
- Undecidable in general
  - Decidable for **monadic** PA
    - All predicates have arity ≤ 1
    - Predicates involving local variables of a single thread

Emptiness Checking

- Determine if an accepting structure is reachable
- Undecidable in general
  - Decidable for **monadic** PA
    - All predicates have arity ≤ 1
    - Predicates involving local variables of a single thread
  - Only consider transitions along *interesting* ids
    - Universe of the current structure and 1 fresh element

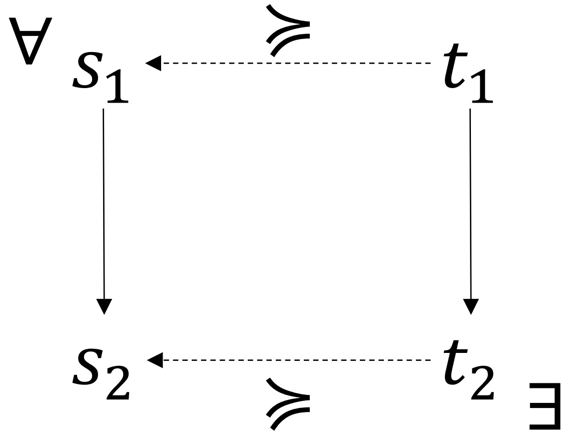
Emptiness Checking

- Determine if an accepting structure is reachable
- Undecidable in general
  - Decidable for **monadic** PA
    - All predicates have arity ≤ 1
    - Predicates involving local variables of a single thread
  - Only consider transitions along *interesting* ids
    - Universe of the current structure and 1 fresh element
  - Use **embeddings** to prune search space (Downward Compatibility)
    - Well structured transition system [Finkel and Schnoebelen. 2001]

Downward Compatibility

A wqo, $\preccurlyeq$, is downward compatible with transition system, $\langle S, \rightarrow \rangle$, if

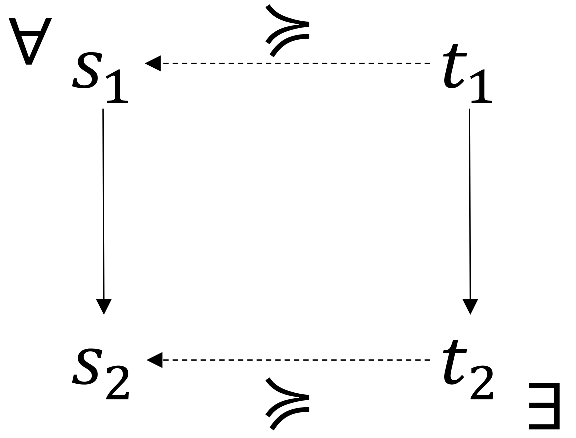
$$\forall\, t_1 \preccurlyeq s_1 \text{ and transition } s_1 \rightarrow s_2 \text{ then } \exists t_2\ s.t.\ t_1 \rightarrow t_2 \text{ and } t_2 \preccurlyeq s_2$$



Downward Compatibility

A wqo, $\preccurlyeq$, is downward compatible with transition system, $\langle S, \rightarrow \rangle$, if

$$\forall\, t_1 \preccurlyeq s_1 \text{ and transition } s_1 \rightarrow s_2 \text{ then } \exists t_2\ s.t.\ t_1 \rightarrow t_2 \text{ and } t_2 \preccurlyeq s_2$$



For PA and embedding if a path from s_1 accepts then a path from t_1 will accept.

Overview

1. Structure Embedding

2. Use in Multi-threaded Verification

3. MatchEmbeds

Match Embeds

Joint work with Zak Kincaid

MatchEmbeds

- Bipartite Graphs
    - Matchings

MatchEmbeds

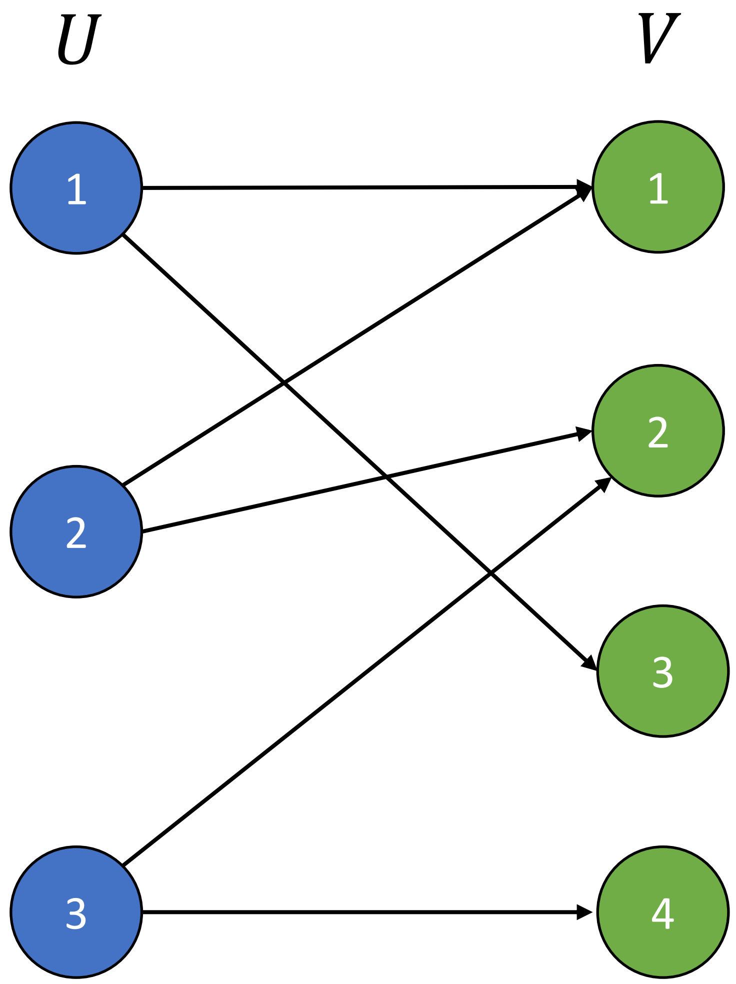
- Bipartite Graphs
  - Matchings
- Monadic Case
  - Reduction to bipartite graph matching

MatchEmbeds

- Bipartite Graphs
  - Matchings
- Monadic Case
  - Reduction to bipartite graph matching
- Generalize bipartite graph matching strategy to general structures
  - Construct bipartite graph
  - Search matchings of graph for an embedding

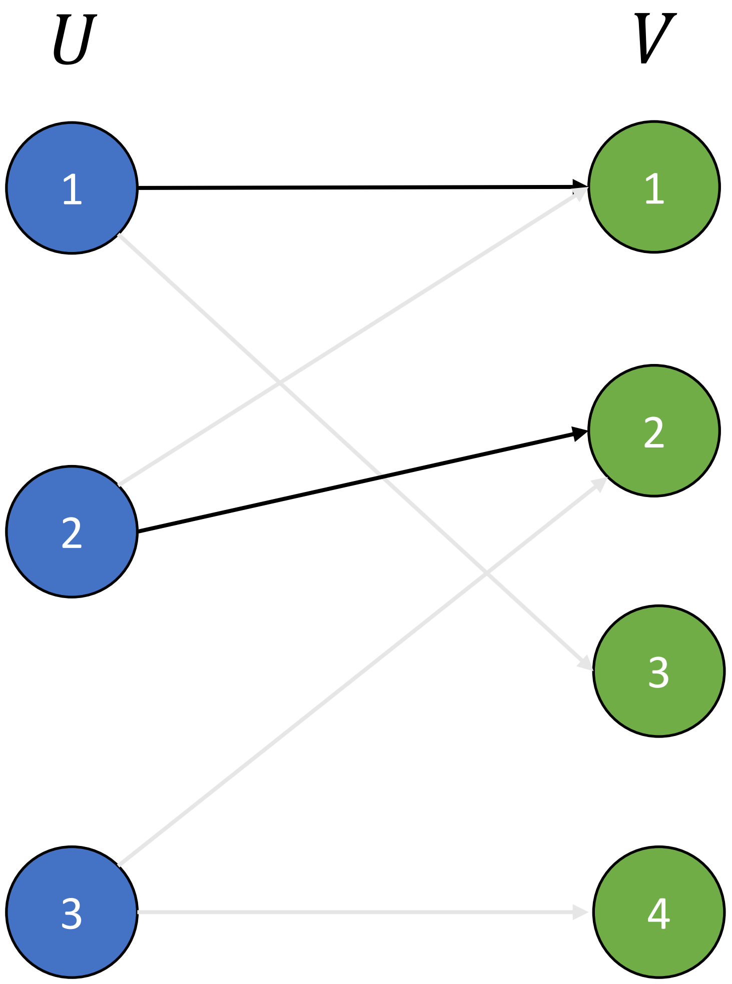
Bipartite Graphs

- Bipartite Graphs, $G = \langle U, V, E \rangle$
  - U and V are disjoint
  - $E \subseteq U \times V$



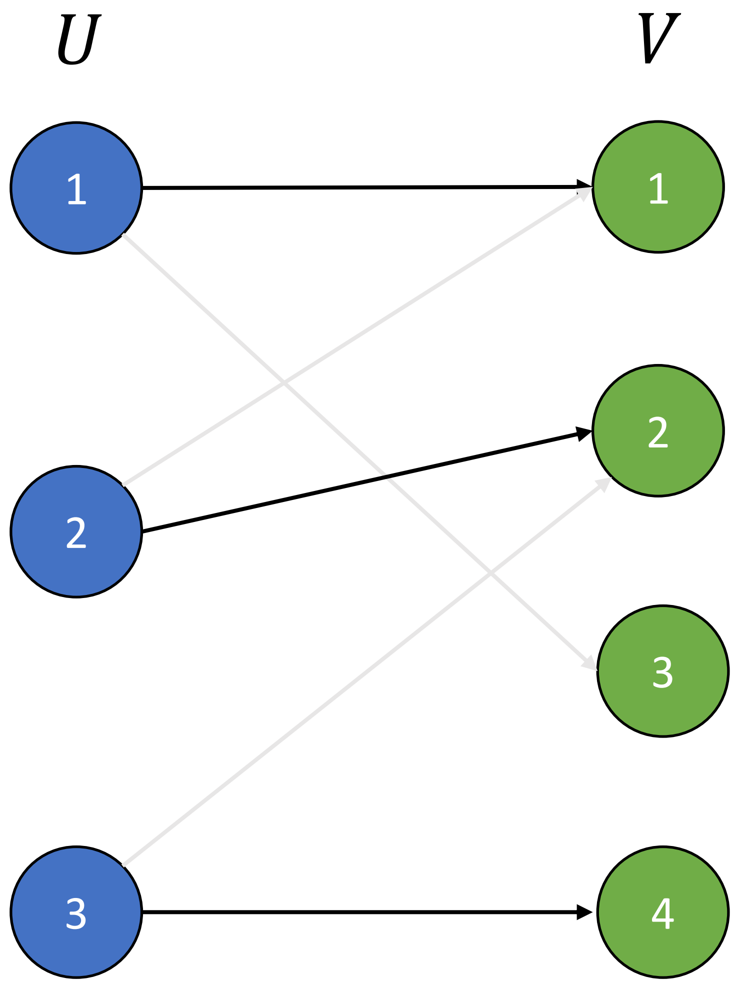
Bipartite Graphs

- Bipartite Graphs, $G = \langle U, V, E \rangle$
  - U and V are disjoint
  - $E \subseteq U \times V$
- Matching, $M \subseteq E$
  - At most one edge contains any vertex
  - $\forall u \in U,\ |\{\langle u, v \rangle \in M\}| \leq 1$
  - $\forall v \in V,\ |\{\langle u, v \rangle \in M\}| \leq 1$



Bipartite Graphs

- Bipartite Graphs, $G = \langle U, V, E \rangle$
  - U and V are disjoint
  - $E \subseteq U \times V$
- Matching, $M \subseteq E$
  - At most one edge contains any vertex
  - $\forall u \in U,\ |\{\langle u, v \rangle \in M\}| \leq 1$
  - $\forall v \in V,\ |\{\langle u, v \rangle \in M\}| \leq 1$
- Total Matching, M
  - M is a matching
  - M covers U ($|M| = |U|$)



Monadic Case

$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$

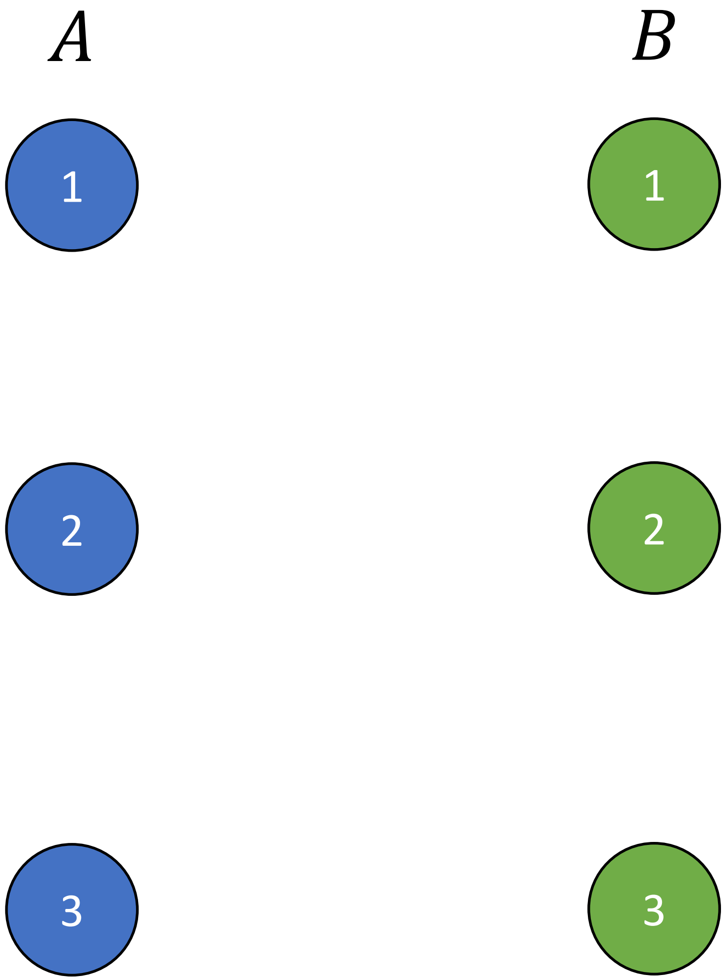
$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$

$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$

$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$



Monadic Case

$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$

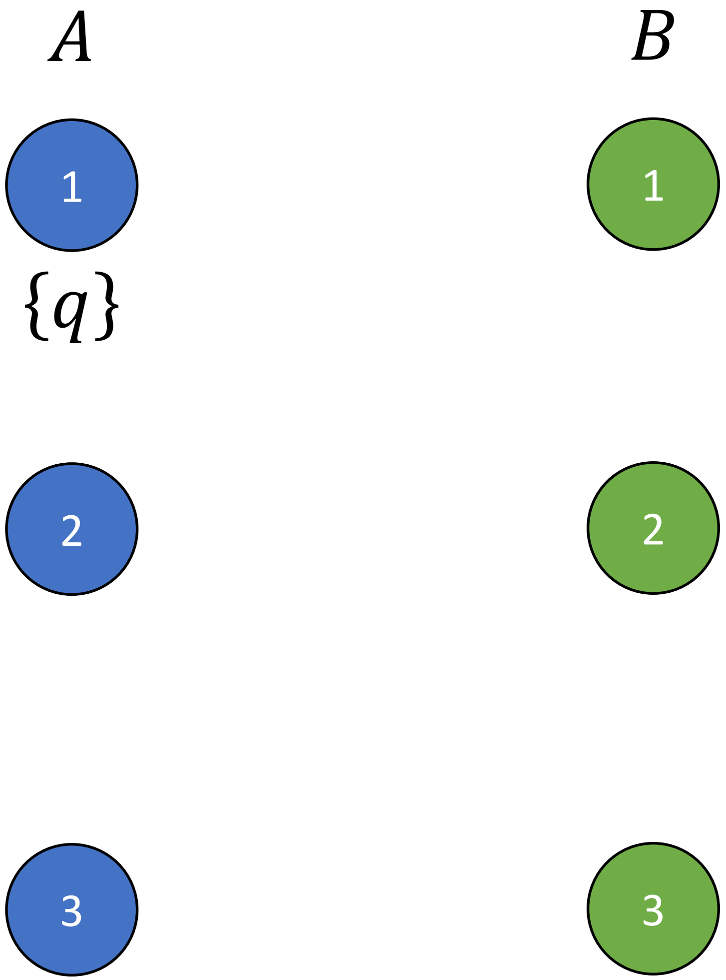
$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$ $\quad sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$

$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$

$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$



Monadic Case

$$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$$

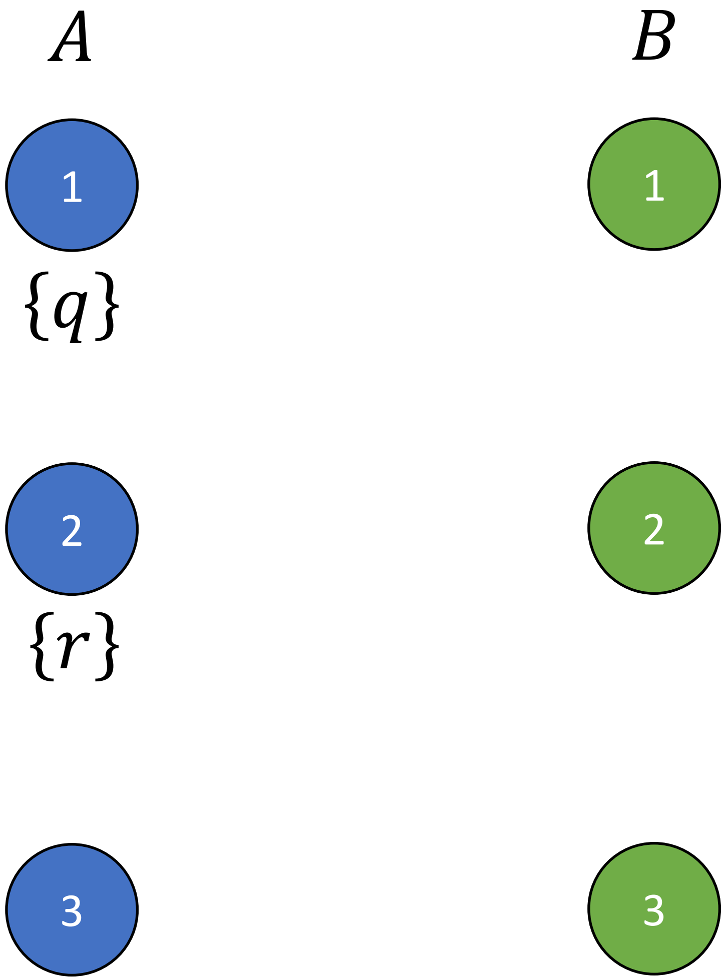
$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$ $\quad$ $sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$ $\quad$ $sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{r\}$

$$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$$

$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$



Monadic Case

$$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q\}$

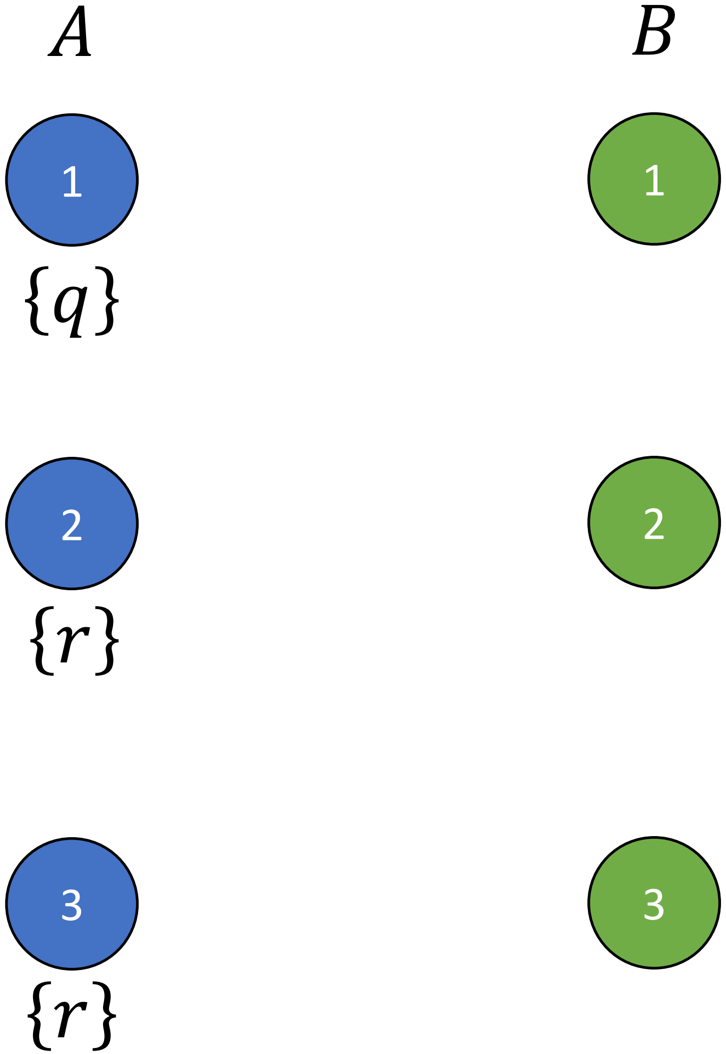
$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{r\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{r\}$

$$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$$

$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$



Monadic Case

$$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{r\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{r\}$

$$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$$

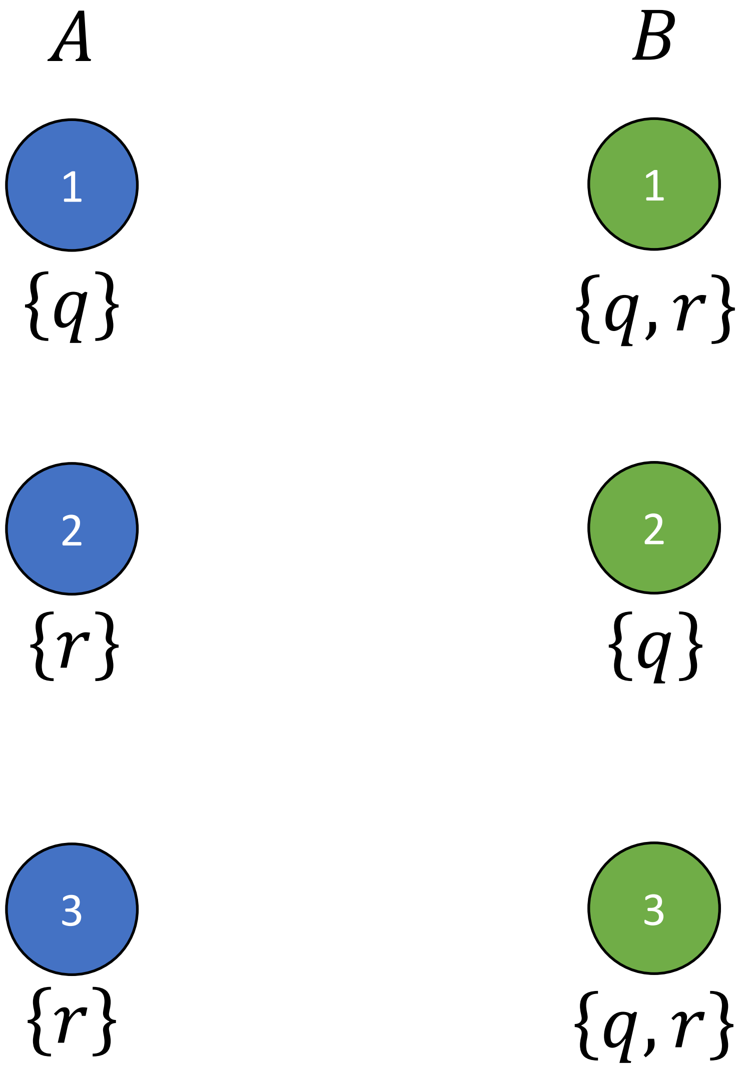
$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q, r\}$

$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{q, r\}$



Monadic Case

$$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{r\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{r\}$

$$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$$

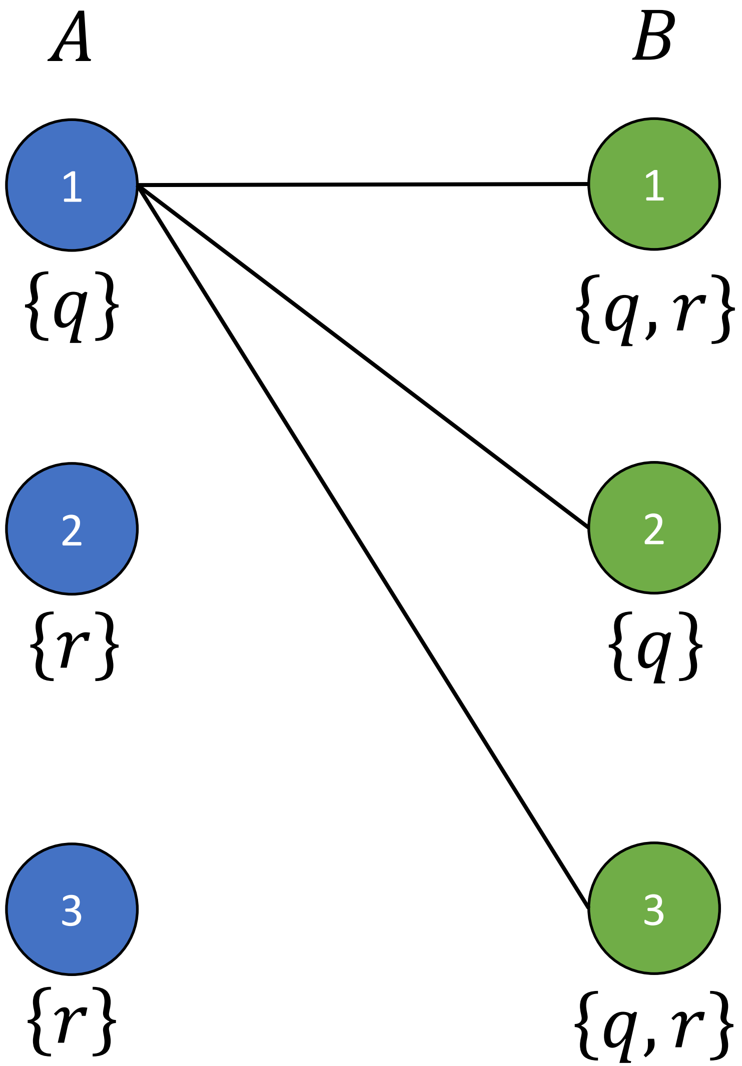
$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q, r\}$

$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{q, r\}$



Monadic Case

$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{r\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{r\}$

$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$

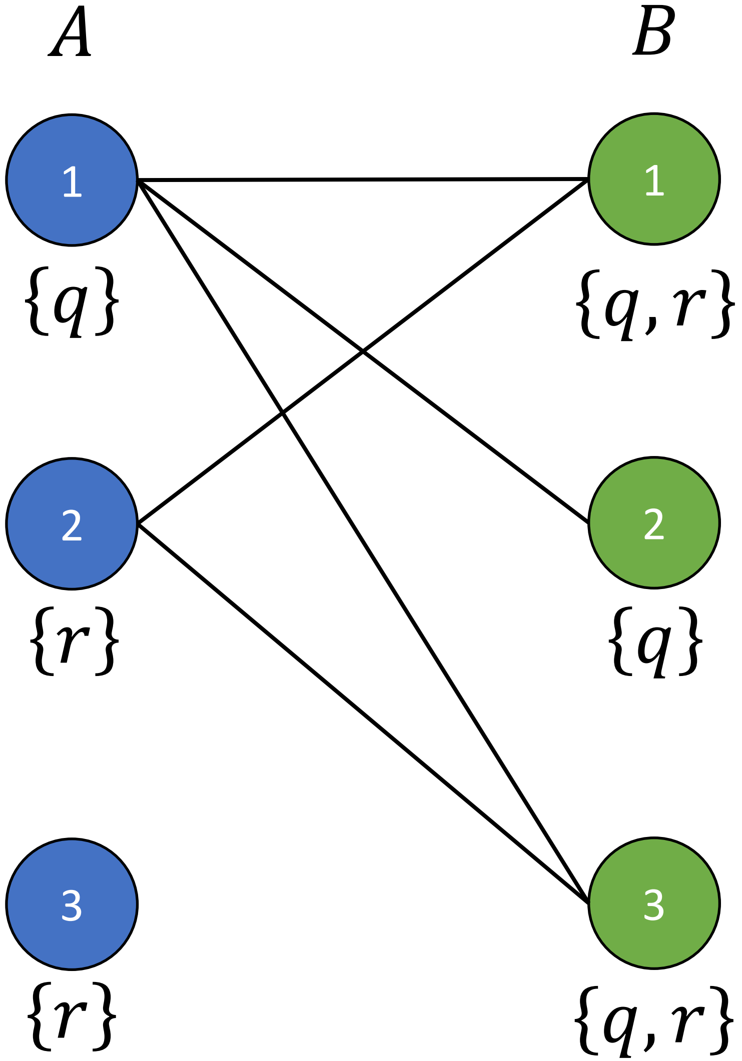
$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q, r\}$

$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{q, r\}$



Monadic Case

$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$ $\quad$ $sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$ $\quad$ $sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{r\}$

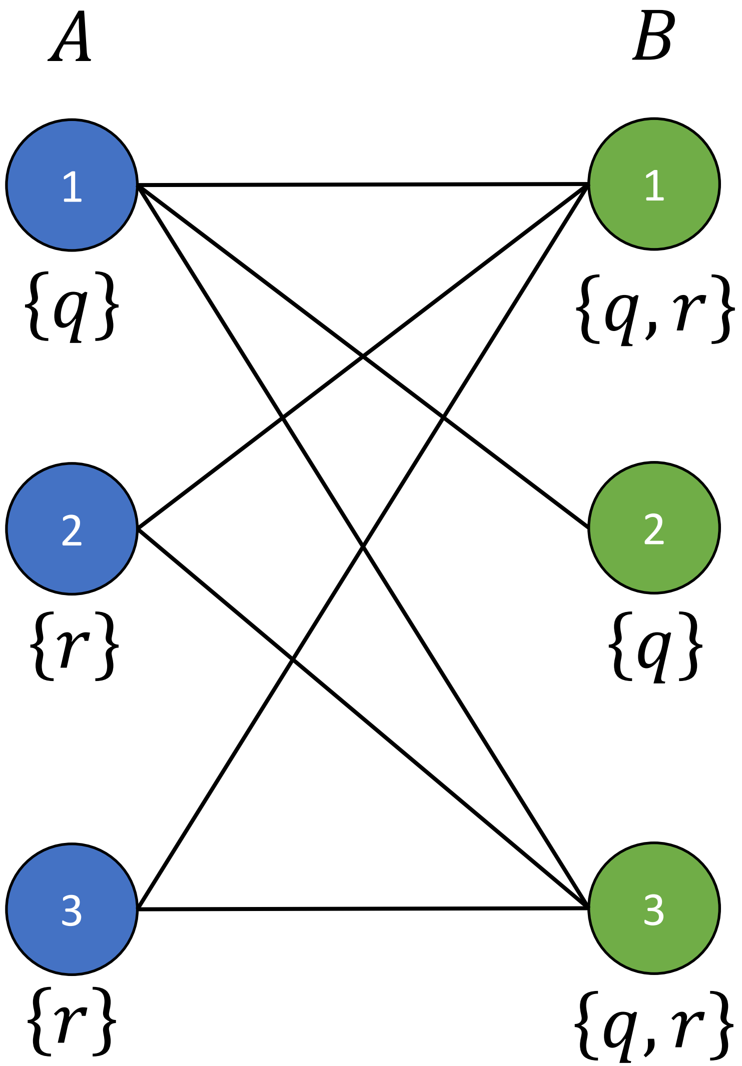
$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{r\}$

$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$

$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$ $\quad$ $sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q, r\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$ $\quad$ $sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{q, r\}$



Monadic Case

$$\mathfrak{A} \overset{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$$

$q^{\mathfrak{A}} \overset{\text{def}}{=} \{1\}$ $\quad$ $sig(\mathfrak{A}, 1) \overset{\text{def}}{=} \{q\}$

$r^{\mathfrak{A}} \overset{\text{def}}{=} \{2,3\}$ $\quad$ $sig(\mathfrak{A}, 2) \overset{\text{def}}{=} \{r\}$

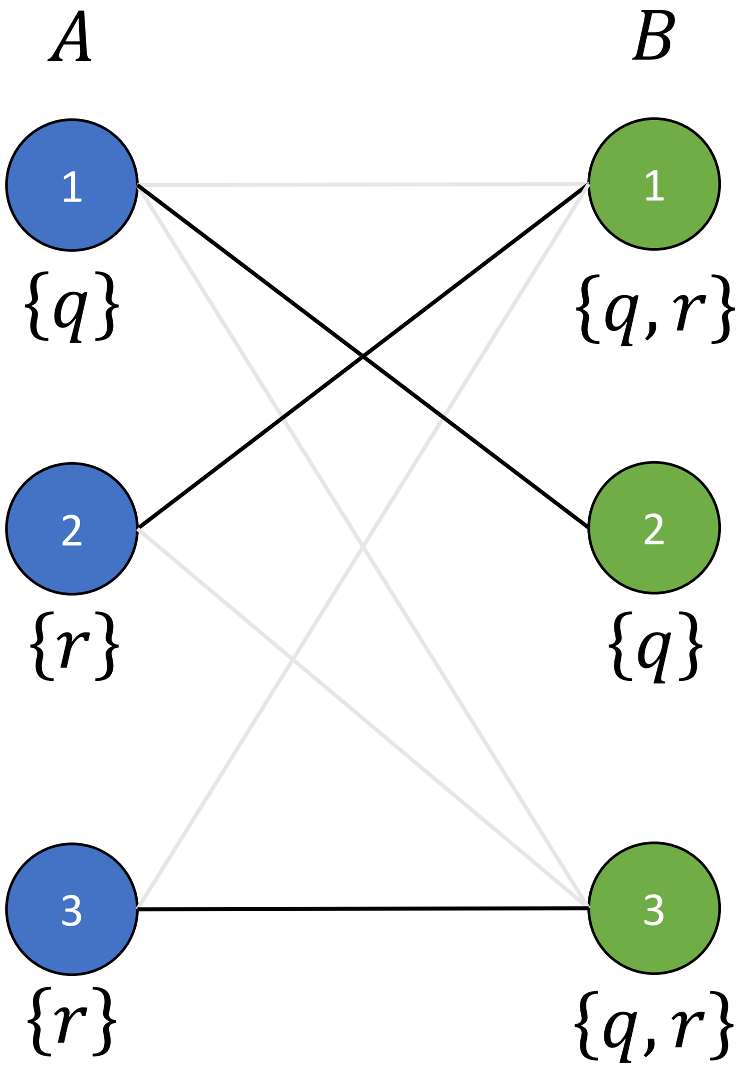
$sig(\mathfrak{A}, 3) \overset{\text{def}}{=} \{r\}$

$$\mathfrak{B} \overset{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$$

$q^{\mathfrak{B}} \overset{\text{def}}{=} \{1,2,3\}$ $\quad$ $sig(\mathfrak{A}, 1) \overset{\text{def}}{=} \{q, r\}$

$r^{\mathfrak{B}} \overset{\text{def}}{=} \{1,3\}$ $\quad$ $sig(\mathfrak{A}, 2) \overset{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 3) \overset{\text{def}}{=} \{q, r\}$



Maximum Matchings[1]

$$M_1 \overset{\text{def}}{=} \left\{ \begin{matrix} \langle 1,2 \rangle, \\ \langle 2,1 \rangle, \\ \langle 3,3 \rangle \end{matrix} \right\}$$

[Hopcroft and Karp. 1973][1]

Monadic Case

$\mathfrak{A} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{A}}, r^{\mathfrak{A}} \rangle$

$q^{\mathfrak{A}} \stackrel{\text{def}}{=} \{1\}$

$r^{\mathfrak{A}} \stackrel{\text{def}}{=} \{2,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{r\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{r\}$

$\mathfrak{B} \stackrel{\text{def}}{=} \langle \{1,2,3\}, q^{\mathfrak{B}}, r^{\mathfrak{B}} \rangle$

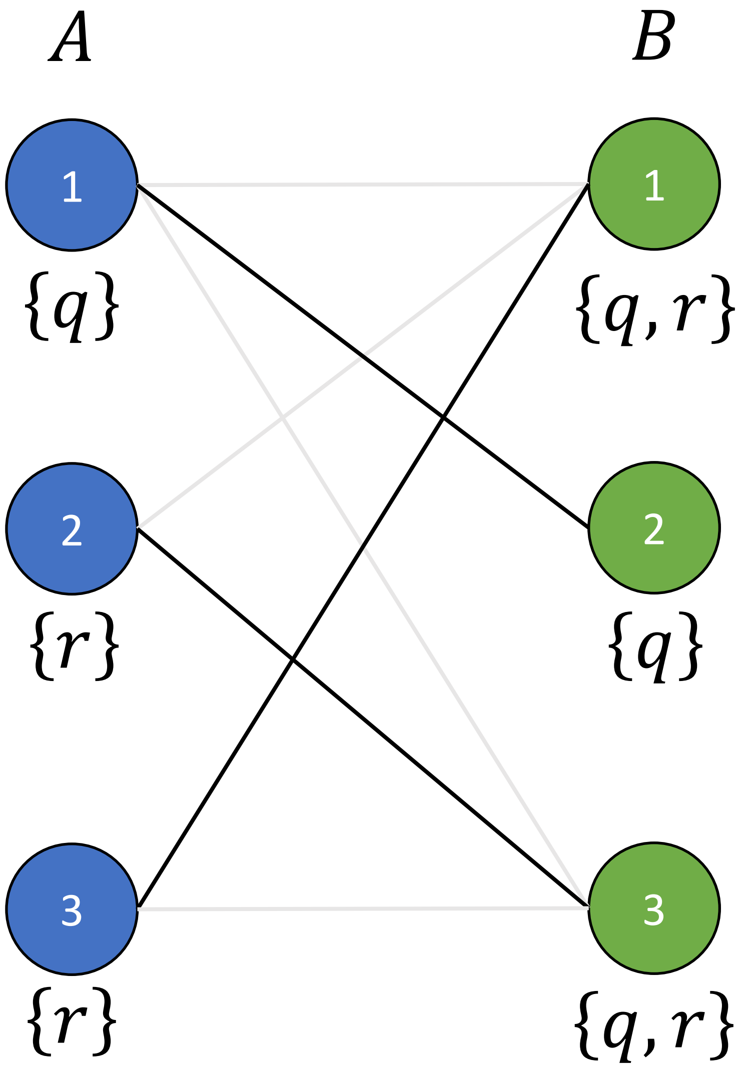
$q^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,2,3\}$

$r^{\mathfrak{B}} \stackrel{\text{def}}{=} \{1,3\}$

$sig(\mathfrak{A}, 1) \stackrel{\text{def}}{=} \{q, r\}$

$sig(\mathfrak{A}, 2) \stackrel{\text{def}}{=} \{q\}$

$sig(\mathfrak{A}, 3) \stackrel{\text{def}}{=} \{q, r\}$



Maximum Matchings[1]

$$M_1 \stackrel{\text{def}}{=} \begin{Bmatrix} \langle 1,2 \rangle, \\ \langle 2,1 \rangle, \\ \langle 3,3 \rangle \end{Bmatrix}$$

$$M_2 \stackrel{\text{def}}{=} \begin{Bmatrix} \langle 1,2 \rangle, \\ \langle 2,3 \rangle, \\ \langle 3,1 \rangle \end{Bmatrix}$$

[Hopcroft and Karp. 1973][1]

Monadic Case

- $\mathfrak{A}$ and $\mathfrak{B}$
  - Structures over common vocabulary
  - Each relation has arity 1

Monadic Case

- $\mathfrak{A}$ and $\mathfrak{B}$
    - Structures over common vocabulary
    - Each relation has arity 1
- **Signature Graph**
    - $sig(\mathfrak{A}, a) \equiv \{q \in Q : q(a) \in \mathfrak{A}\}$

Monadic Case

- $\mathfrak{A}$ and $\mathfrak{B}$
  - Structures over common vocabulary
  - Each relation has arity 1
- **Signature Graph**
  - $sig(\mathfrak{A}, a) \stackrel{\text{def}}{=} \left\{q \in Q \colon \exists \left\langle a_1, \ldots, a_{ar(q)} \right\rangle \in q^{\mathfrak{A}}. \exists i. a = a_i \right\}$

Monadic Case

- $\mathfrak{A}$ and $\mathfrak{B}$
  - Structures over common vocabulary
  - Each relation has arity 1
- **Signature Graph**
  - $sig(\mathfrak{A}, a) \overset{\text{def}}{=} \left\{q \in Q : \exists \left\langle a_1, \dots, a_{ar(q)} \right\rangle \in q^{\mathfrak{A}}. \exists i. a = a_i \right\}$
  - $Sig(\mathfrak{A}, \mathfrak{B}) \overset{\text{def}}{=} G(A, B, E)$
    - $E \overset{\text{def}}{=} \{\langle a, b\rangle \in A \times B : sig(\mathfrak{A}, a) \subseteq sig(\mathfrak{B}, b)\}$

Monadic Case

- $\mathfrak{A}$ and $\mathfrak{B}$
  - Structures over common vocabulary
  - Each relation has arity 1
- **Signature Graph**
  - $sig(\mathfrak{A}, a) \stackrel{\text{def}}{=} \left\{q \in Q: \exists \left\langle a_1, \dots, a_{ar(q)} \right\rangle \in q^{\mathfrak{A}}. \exists i. a = a_i\right\}$
  - $Sig(\mathfrak{A}, \mathfrak{B}) \stackrel{\text{def}}{=} G(A, B, E)$
    - $E \stackrel{\text{def}}{=} \{\langle a, b\rangle \in A \times B: sig(\mathfrak{A}, a) \subseteq sig(\mathfrak{B}, b)\}$
  - $M \subseteq E$ is a total matching on A iff f_M is a structure embedding

Monadic Case

- $\mathfrak{A}$ and $\mathfrak{B}$
  - Structures over common vocabulary
  - Each relation has arity 1
- **Signature Graph**
  - $sig(\mathfrak{A}, a) \stackrel{\text{def}}{=} \left\{q \in Q : \exists \langle a_1, \dots, a_{ar(q)} \rangle \in q^{\mathfrak{A}}. \exists i. a = a_i\right\}$
  - $Sig(\mathfrak{A}, \mathfrak{B}) \stackrel{\text{def}}{=} G(A, B, E)$
    - $E \stackrel{\text{def}}{=} \{\langle a, b \rangle \in A \times B : sig(\mathfrak{A}, a) \subseteq sig(\mathfrak{B}, b)\}$
  - $M \subseteq E$ is a total matching on A iff f_M is a structure embedding
- Structure embedding takes $O(|A||B|\sqrt{|A| + |B|})$ [Hopcroft and Karp. 1973]

Match Embeds

- Inspired by monadic reduction to bipartite graph matching

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings
  1. Remove inconsistent edges from graph

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings
  1. Remove inconsistent edges from graph
  2. Compute maximum matching

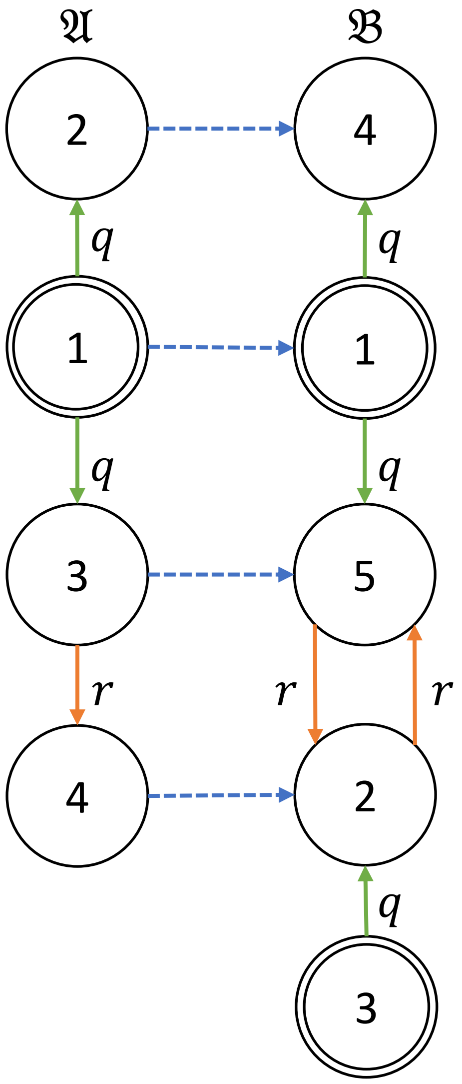
Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings
  1. Remove inconsistent edges from graph
  2. Compute maximum matching
  3. Check for conflicts

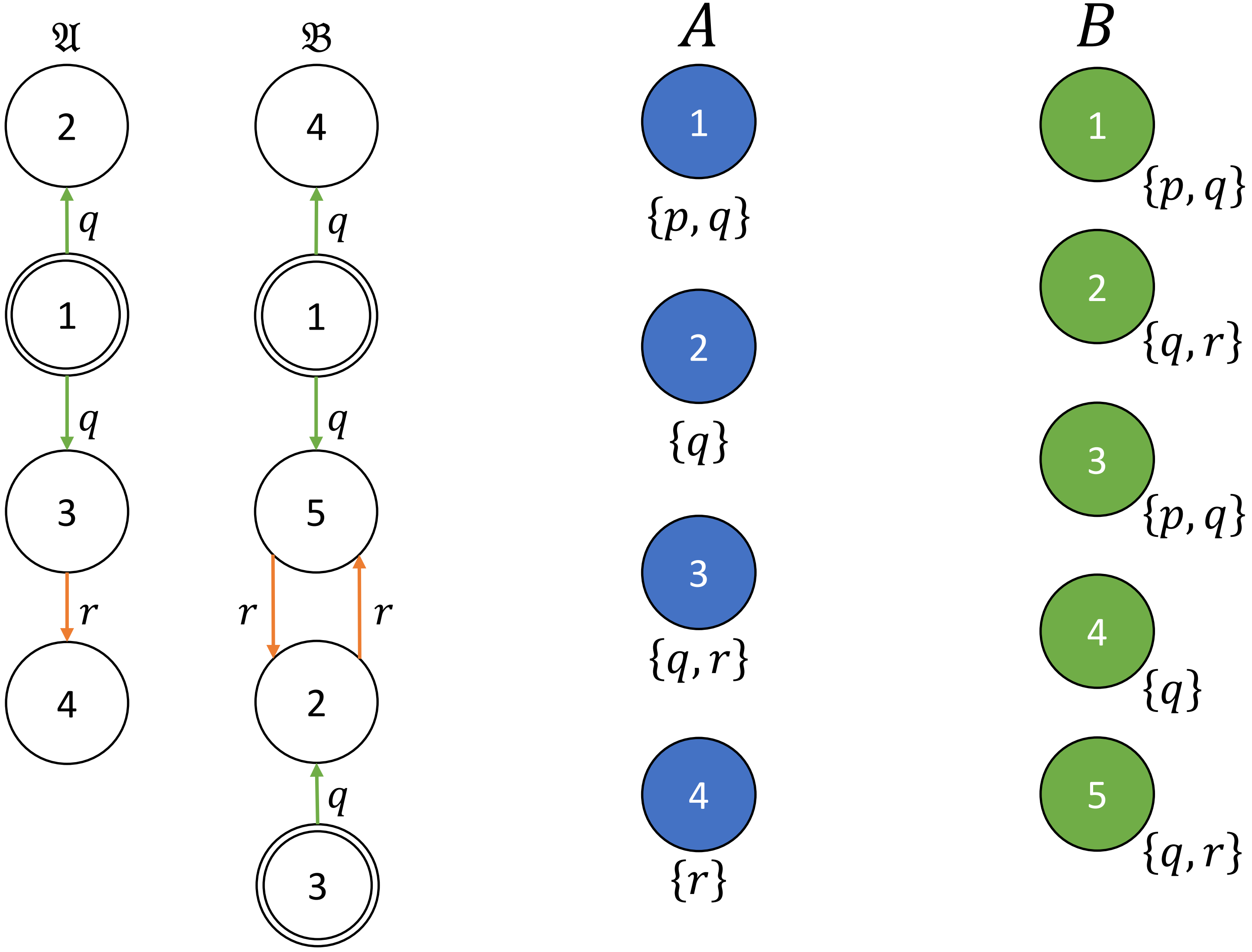
Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings
  1. Remove inconsistent edges from graph
  2. Compute maximum matching
  3. Check for conflicts
  4. Decide on edges in matching and recurse

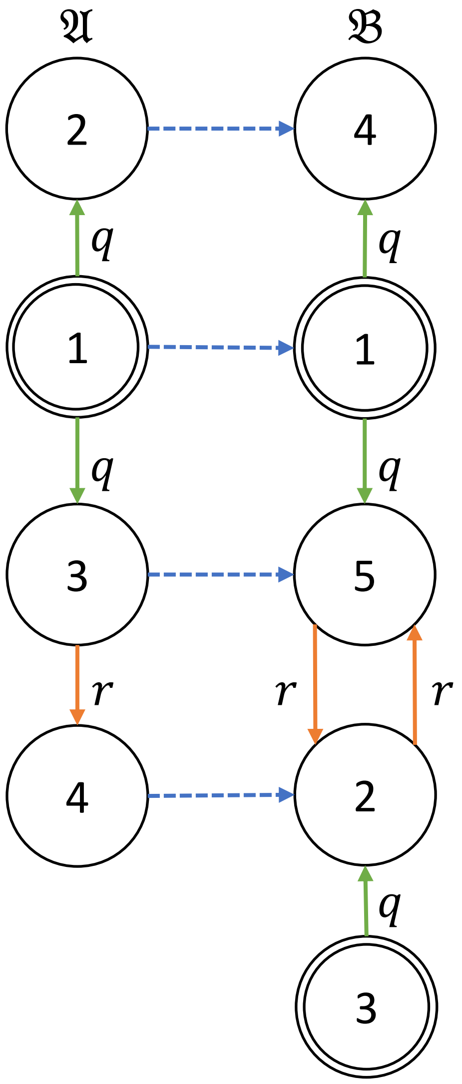
General Case

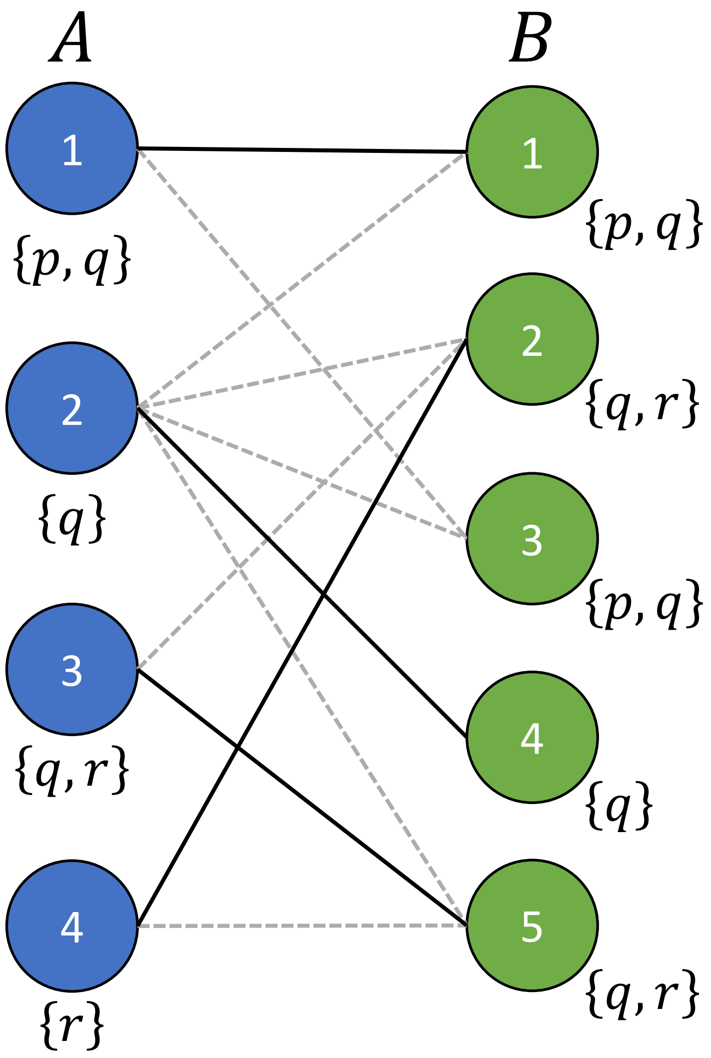


General Case



General Case





$M_1 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

$\boldsymbol{M_2 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}}$

$M_3 \stackrel{\text{def}}{=} \{\langle 1,3\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

$M_4 \stackrel{\text{def}}{=} \{\langle 1,3\rangle, \langle 2,4\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}$

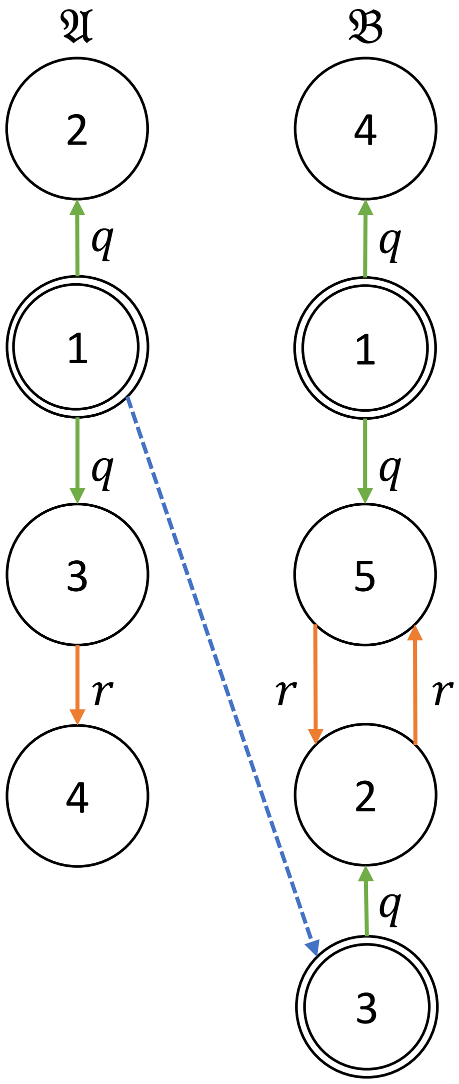
$M_5 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,3\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

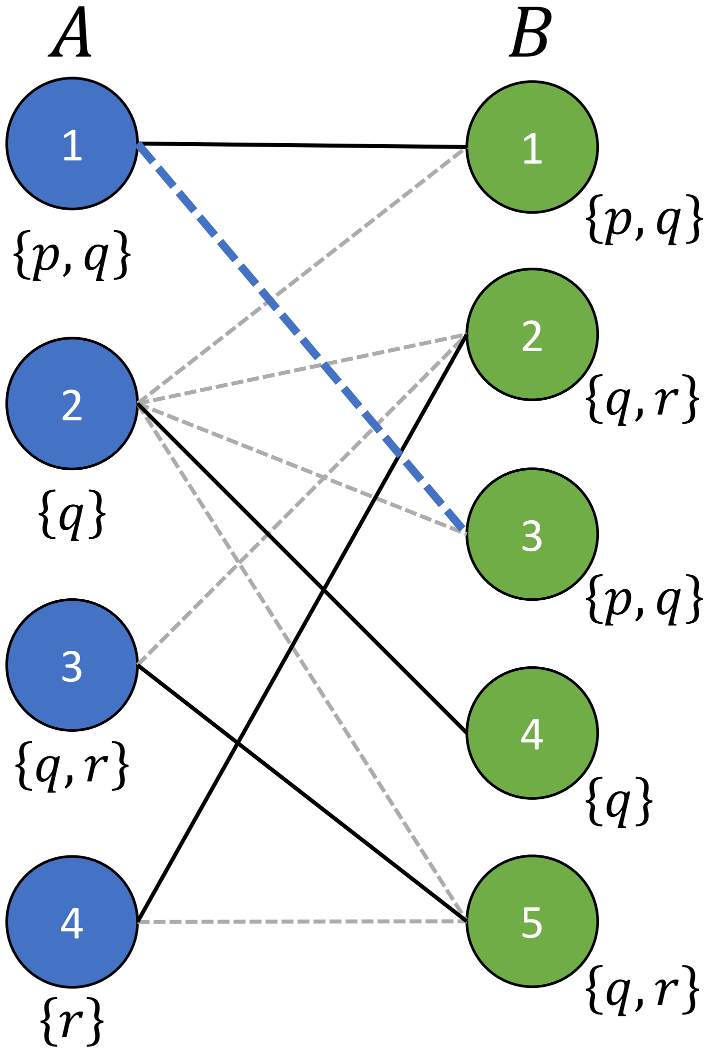
$M_6 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,3\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}$

$M_7 \stackrel{\text{def}}{=} \{\langle 1,3\rangle, \langle 2,1\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

$M_8 \stackrel{\text{def}}{=} \{\langle 1,3\rangle, \langle 2,1\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}$

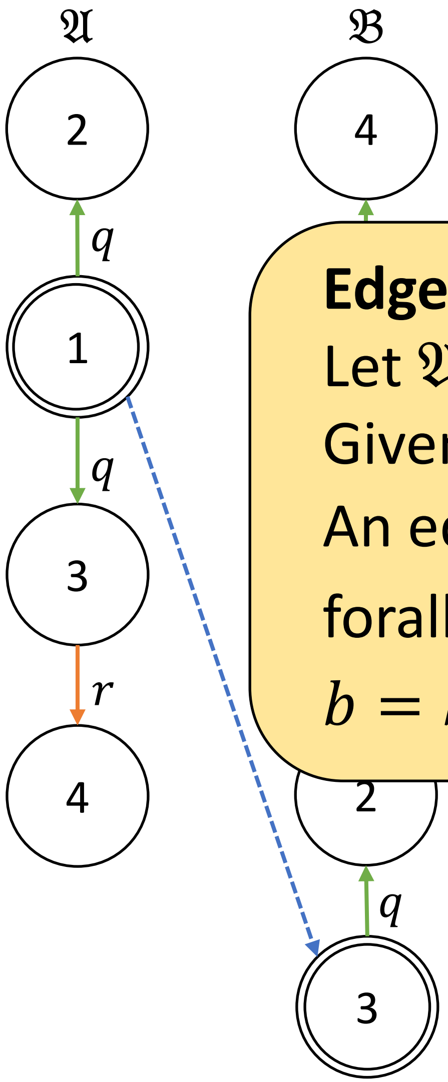
Consistency

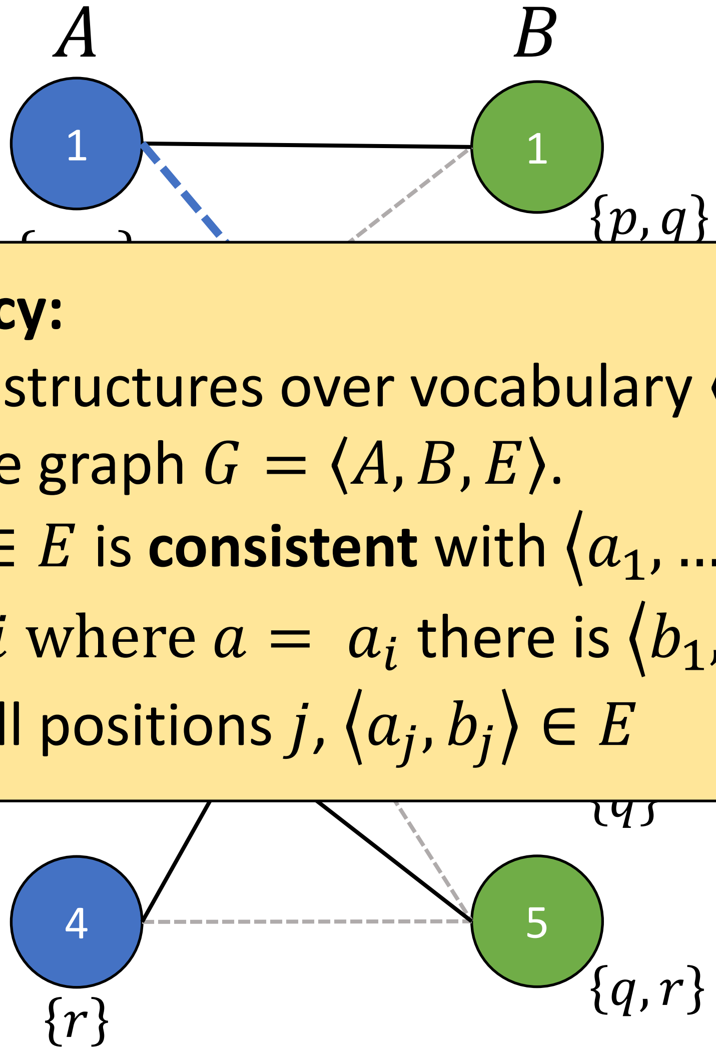




Consider edge ⟨1,3⟩:

Consistency



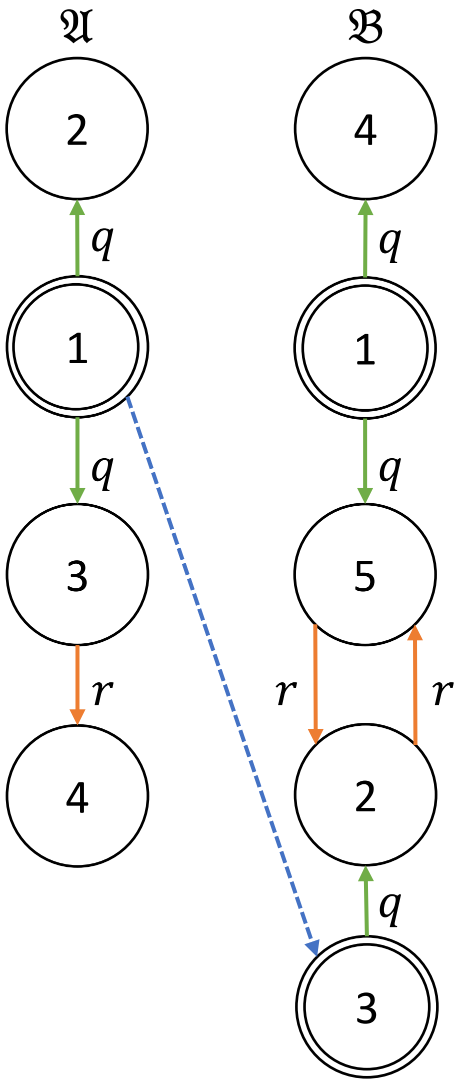


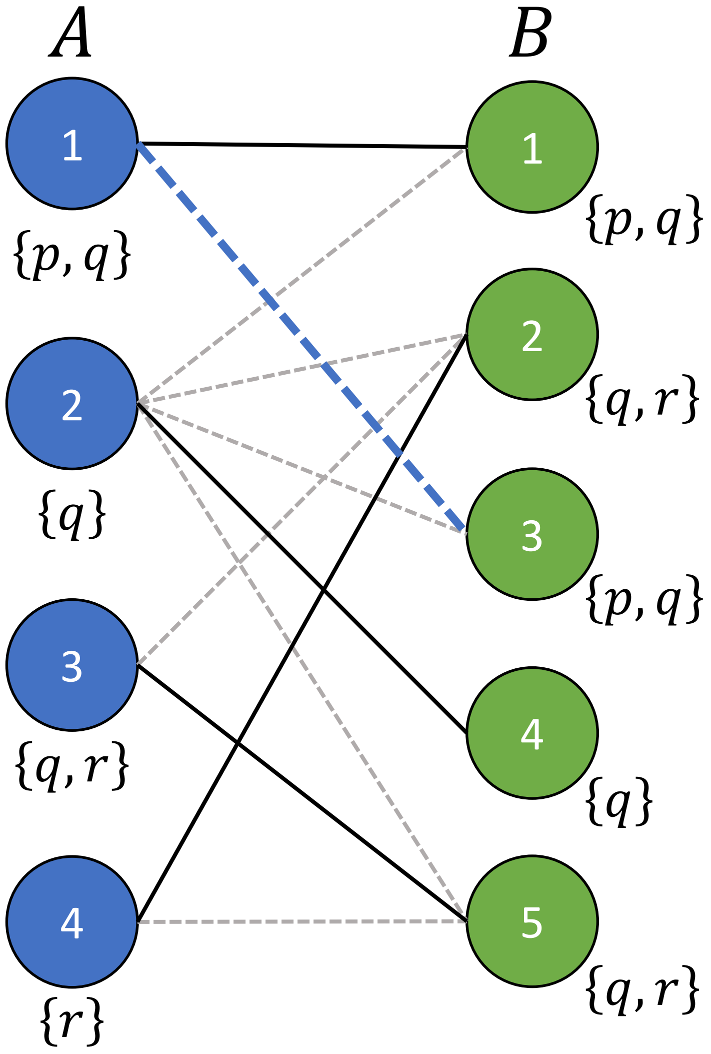
Consider edge $\langle 1,3 \rangle$:

Edge Consistency:

Let $\mathfrak{A}$ and $\mathfrak{B}$ be structures over vocabulary $\langle Q, ar \rangle$.
Given a bipartite graph $G = \langle A, B, E \rangle$.
An edge $\langle a, b \rangle \in E$ is **consistent** with $\langle a_1, \dots, a_{ar(q)} \rangle \in q^{\mathfrak{A}}$ when forall positions i where $a = a_i$ there is $\langle b_1, \dots, b_{ar(q)} \rangle \in q^{\mathfrak{B}}$ s.t. $b = b_i$ and forall positions j, $\langle a_j, b_j \rangle \in E$

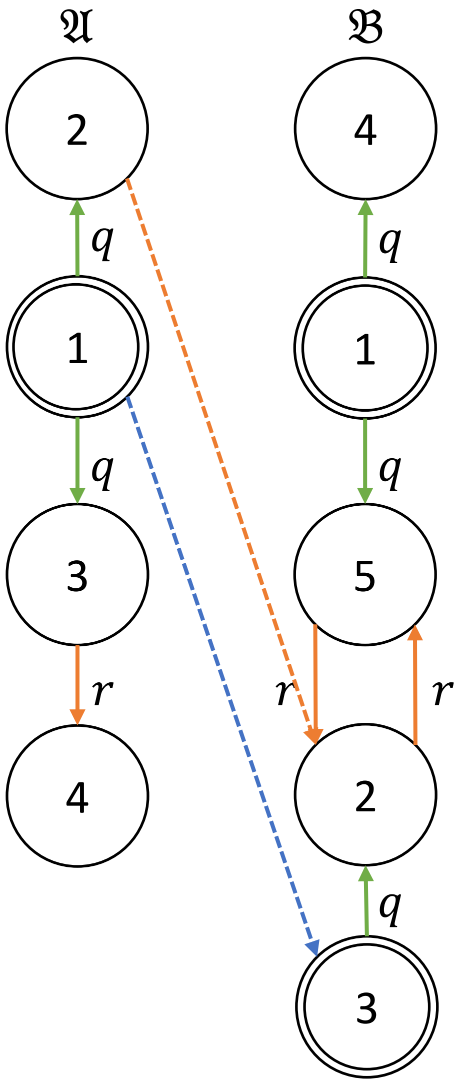
Consistency

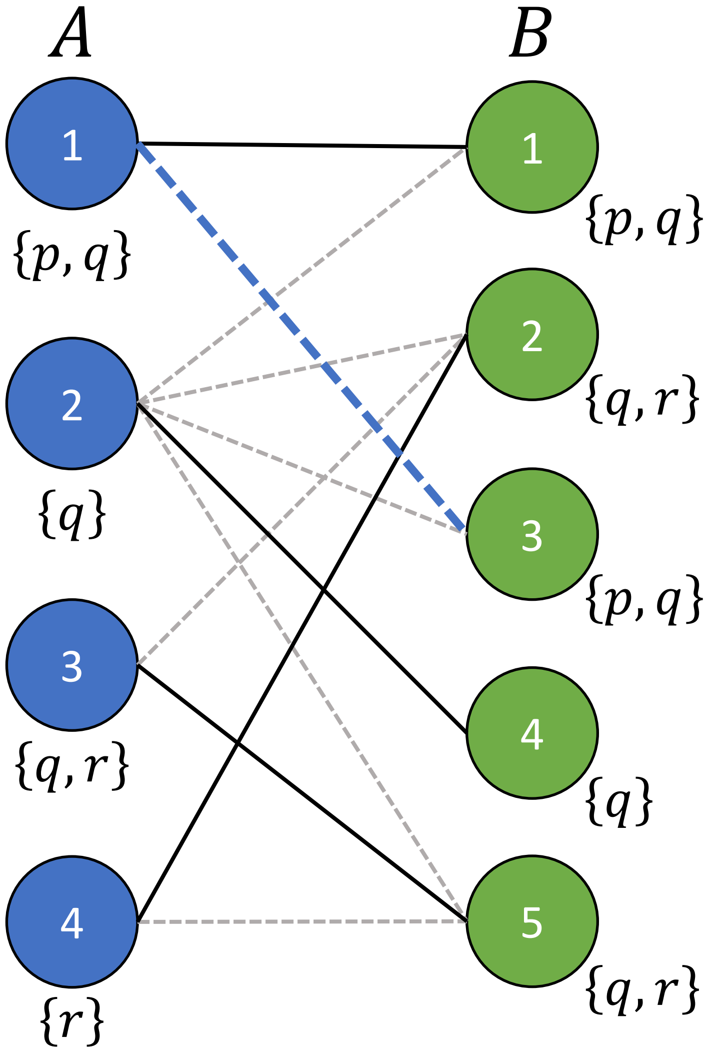




Consider edge ⟨1,3⟩:

Consistency

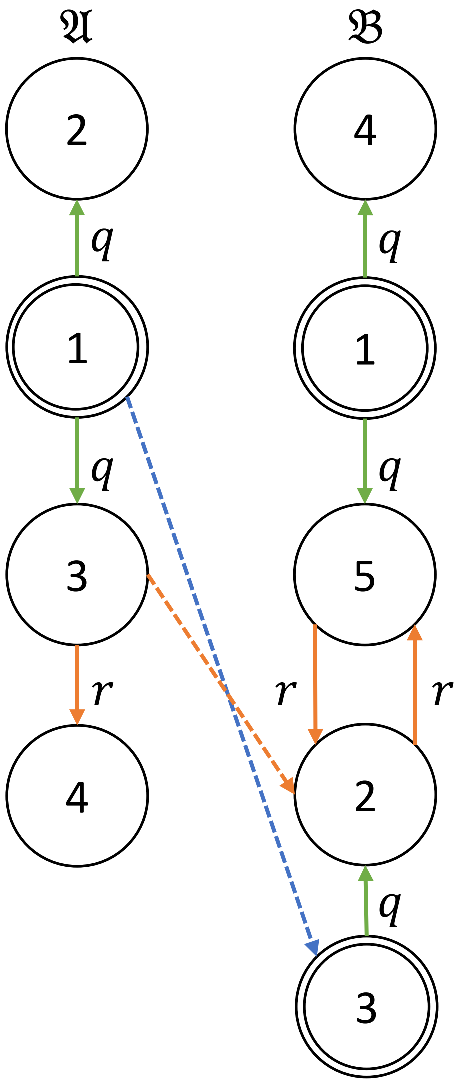


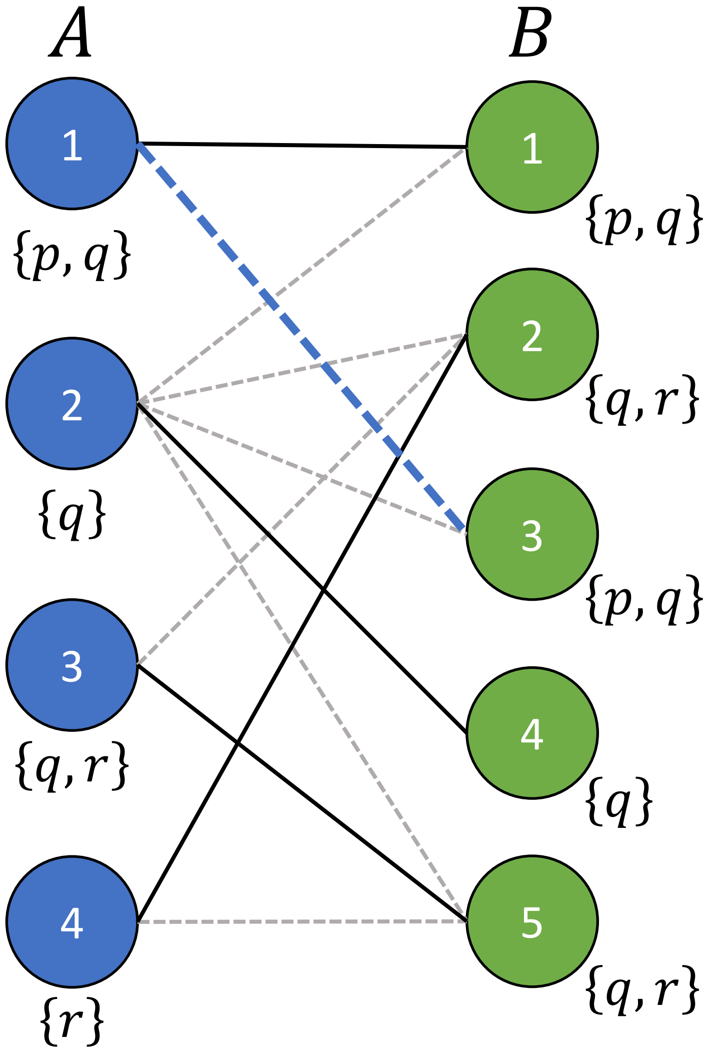


Consider edge ⟨1,3⟩:

- Consistent with $q(1,2)$
  - $\exists q(3,2) \in \mathfrak{B} \wedge \langle 2,2\rangle \in G$

Consistency

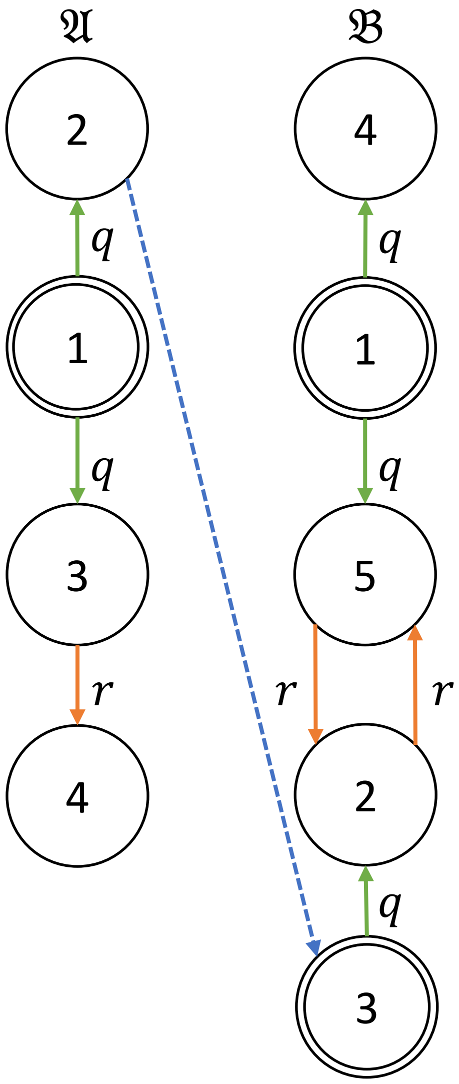


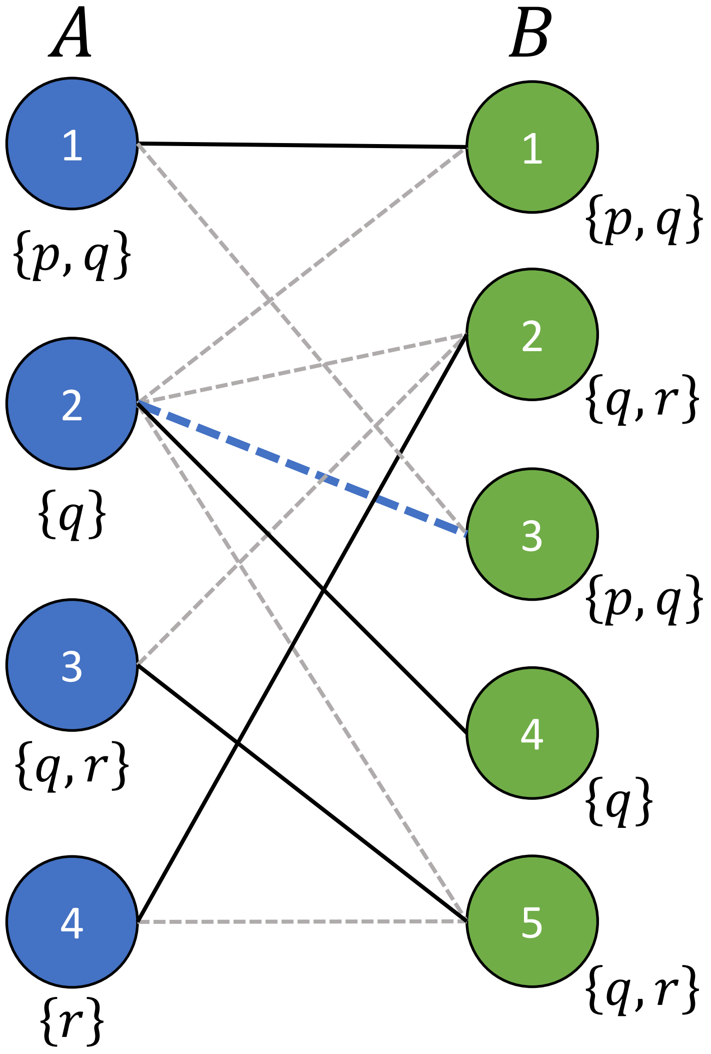


Consider edge $\langle 1,3 \rangle$:

- Consistent with $q(1,2)$
  - $\exists q(3,2) \in \mathfrak{B} \wedge \langle 2,2 \rangle \in G$
- Consistent with $q(1,3)$
  - $\exists q(3,2) \in \mathfrak{B} \wedge \langle 3,2 \rangle \in G$

Consistency

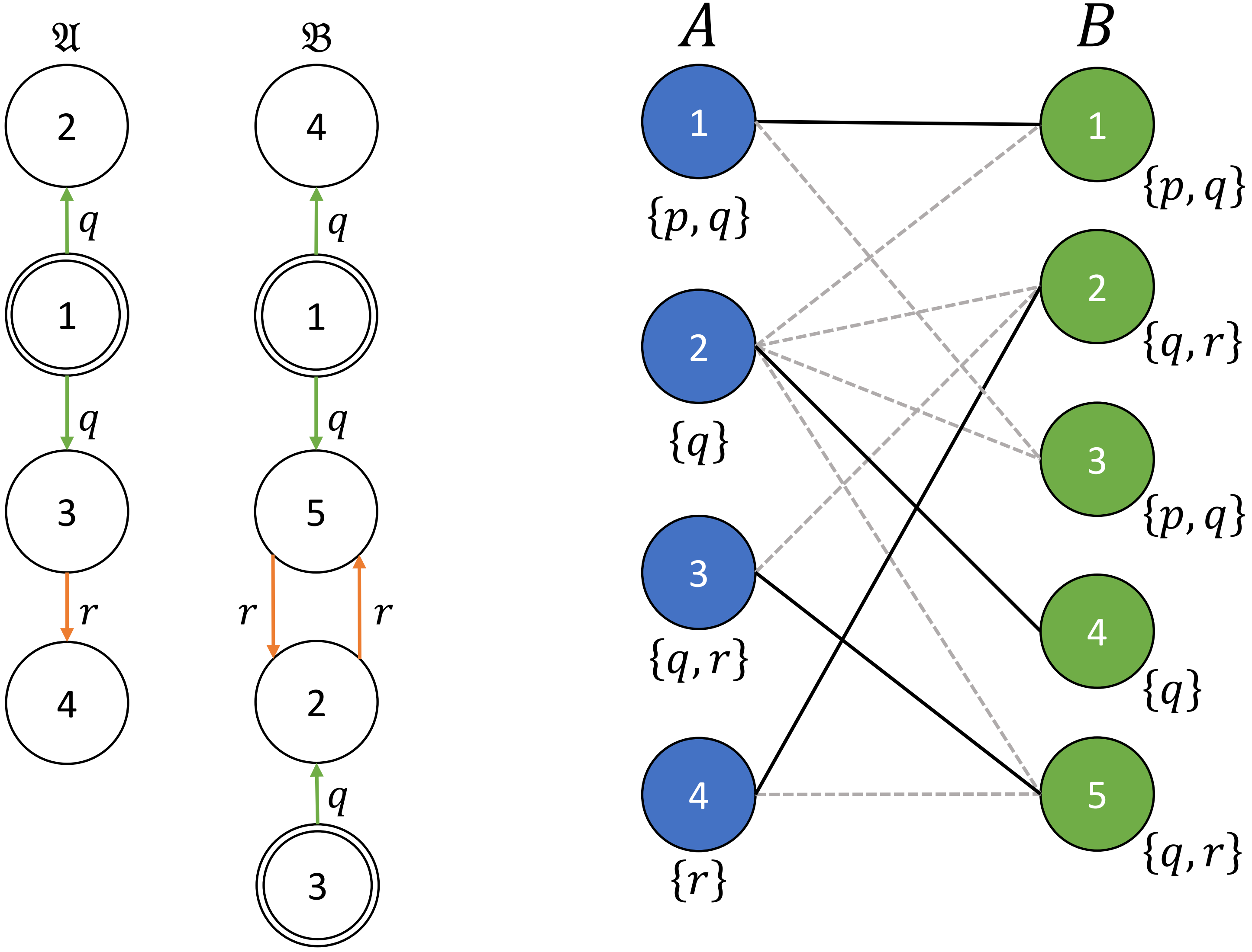




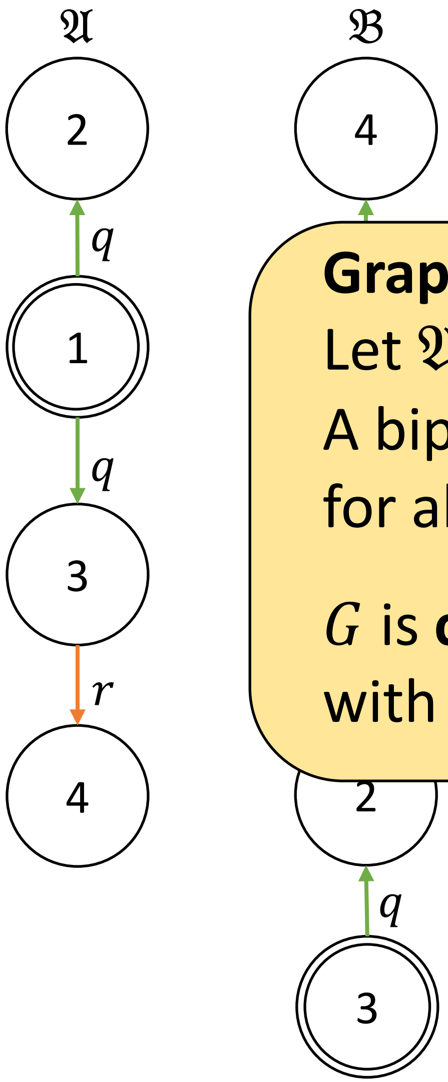
Consider edge $\langle 2,3 \rangle$:

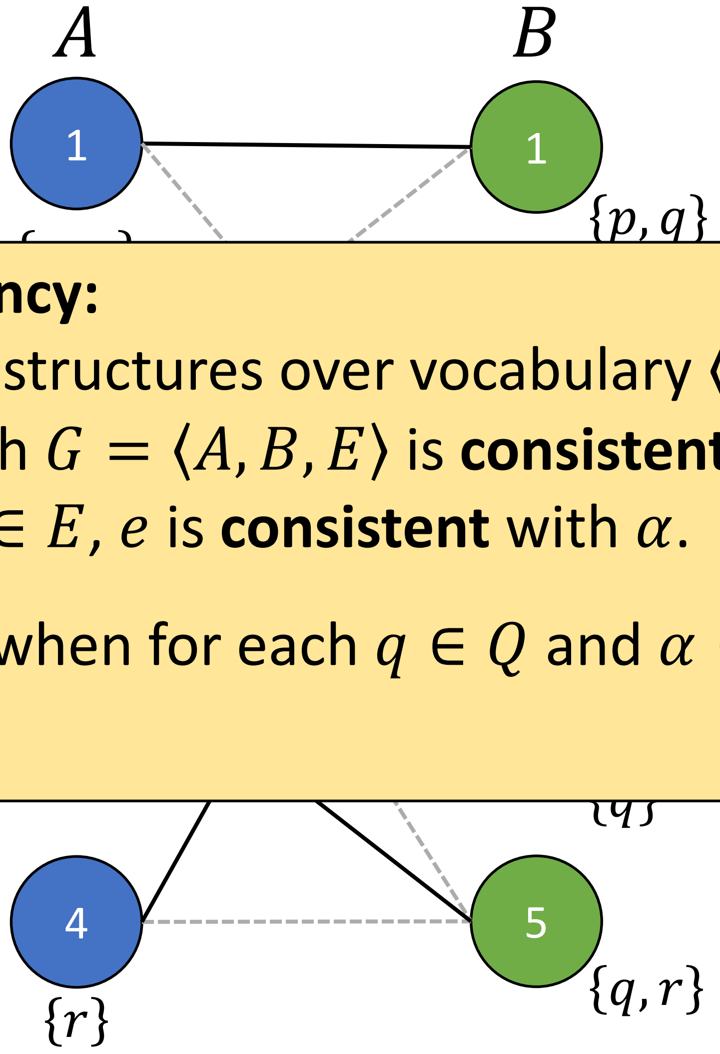
- Inconsistent with $q(1,2)$
  - $\nexists q(*, 3) \in \mathfrak{B}$

Maximum Consistent Sub-Graph



Maximum Consistent Sub-Graph

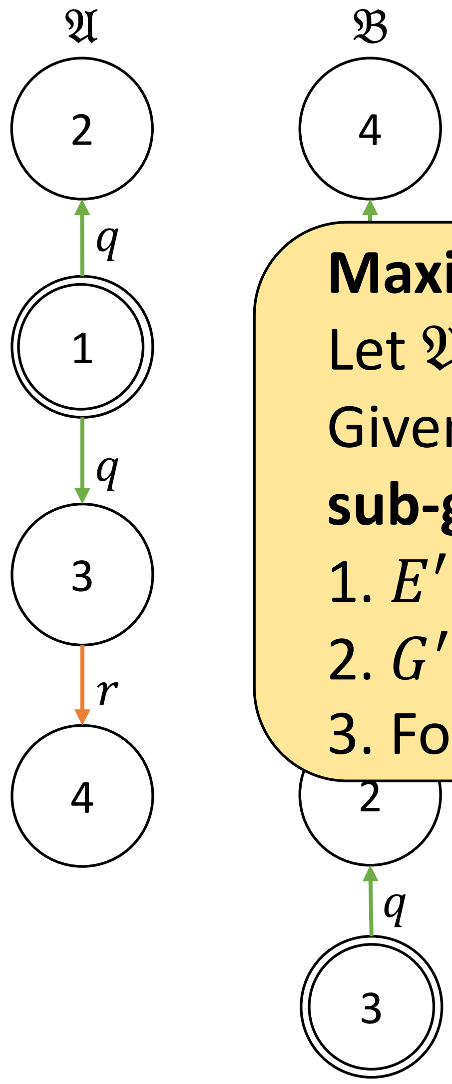


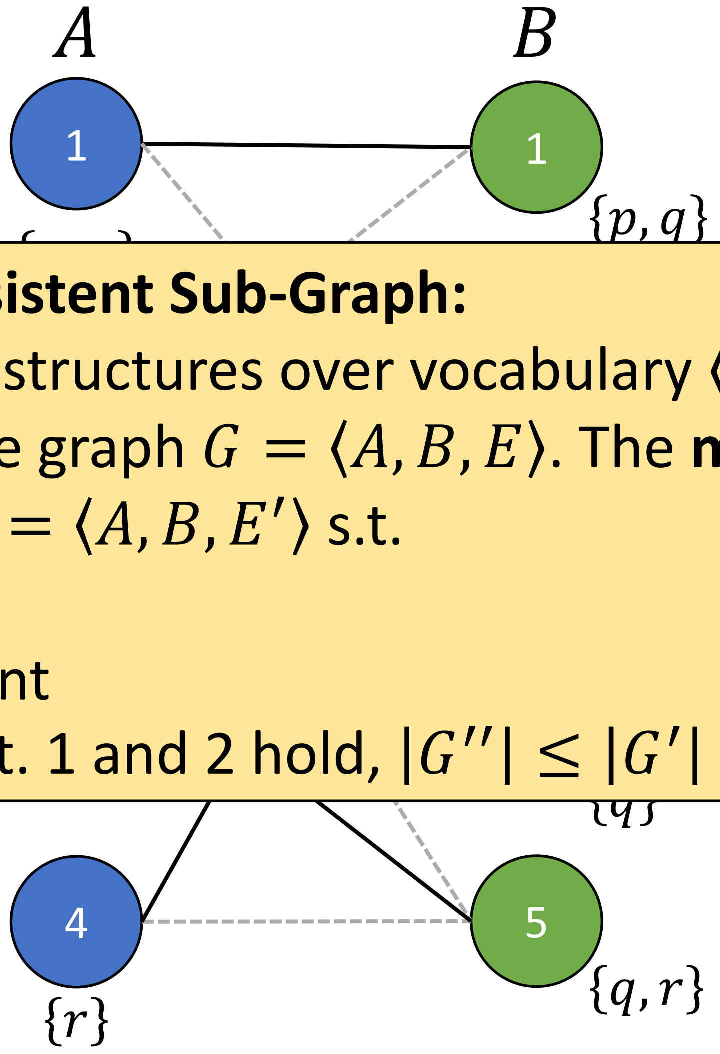


Graph Consistency:
Let $\mathfrak{A}$ and $\mathfrak{B}$ be structures over vocabulary $\langle Q, ar \rangle$.
A bipartite graph $G = \langle A, B, E \rangle$ is **consistent** with $\alpha \in q^{\mathfrak{A}}$ when for all edges, $e \in E$, e is **consistent** with α.

G is **consistent** when for each $q \in Q$ and $\alpha \in q^{\mathfrak{A}}$, G is **consistent** with α.

Maximum Consistent Sub-Graph

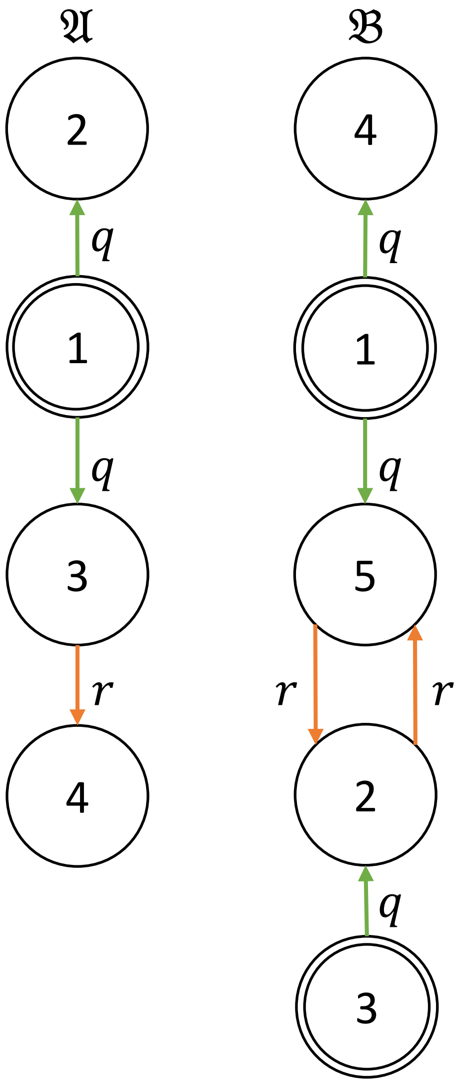


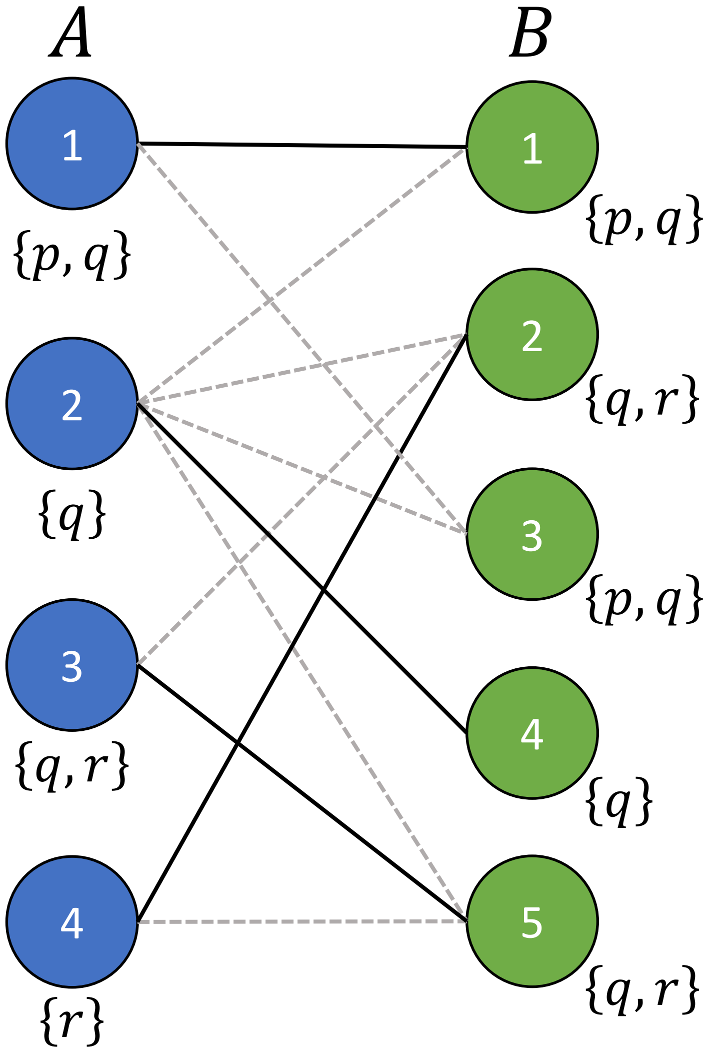


Maximum Consistent Sub-Graph:
Let $\mathfrak{A}$ and $\mathfrak{B}$ be structures over vocabulary $\langle Q, ar\rangle$.
Given a bipartite graph $G = \langle A, B, E\rangle$. The **maximum consistent sub-graph** is $G' = \langle A, B, E'\rangle$ s.t.

1. $E' \subseteq E$
2. G' is consistent
3. For any G'' s.t. 1 and 2 hold, $|G''| \leq |G'|$

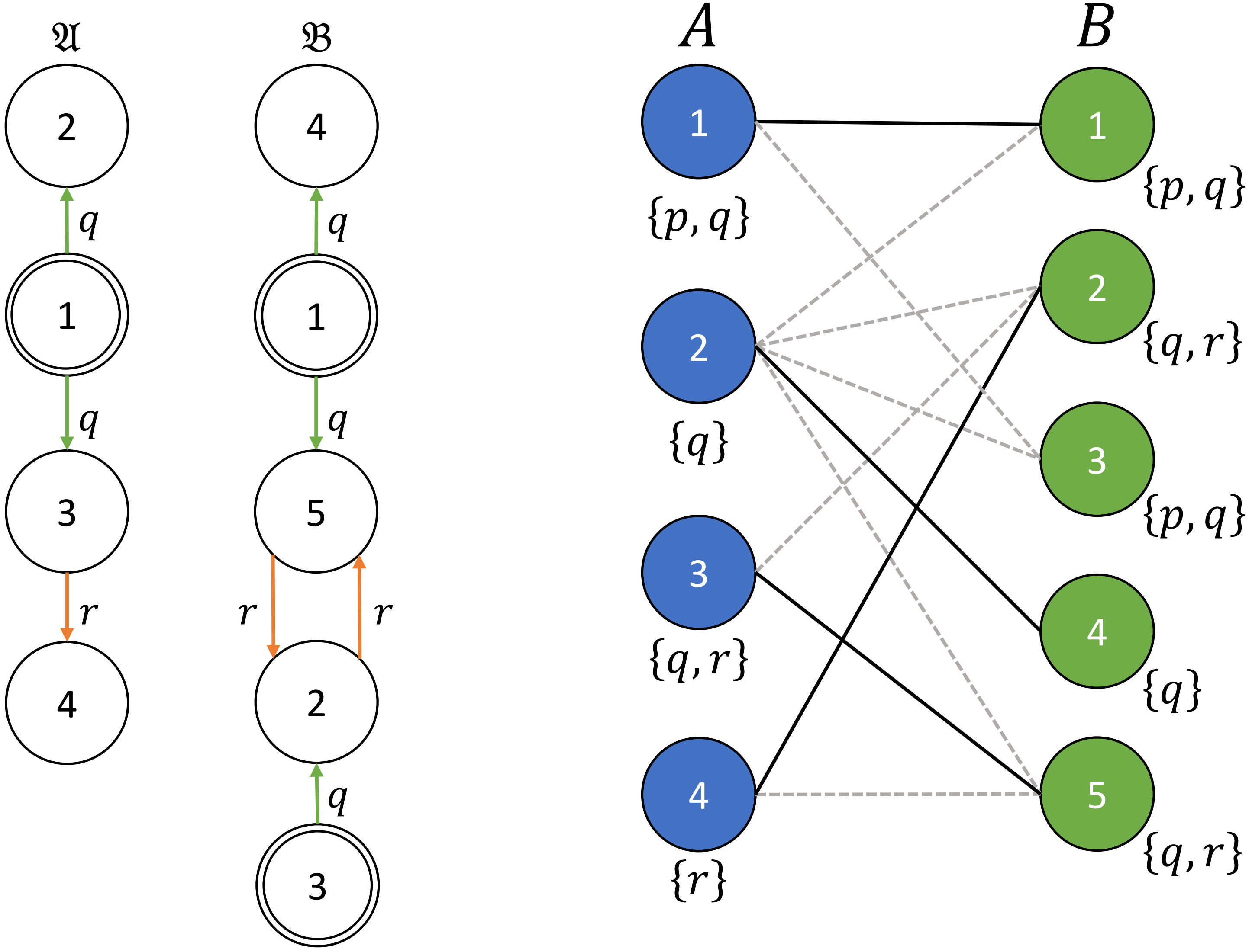
Maximum Consistent Sub-Graph





Goals:

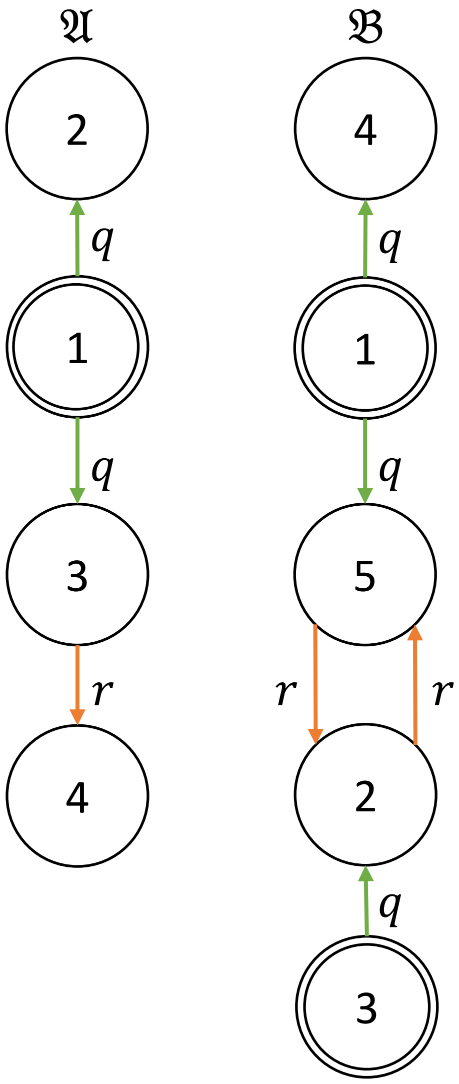
Maximum Consistent Sub-Graph

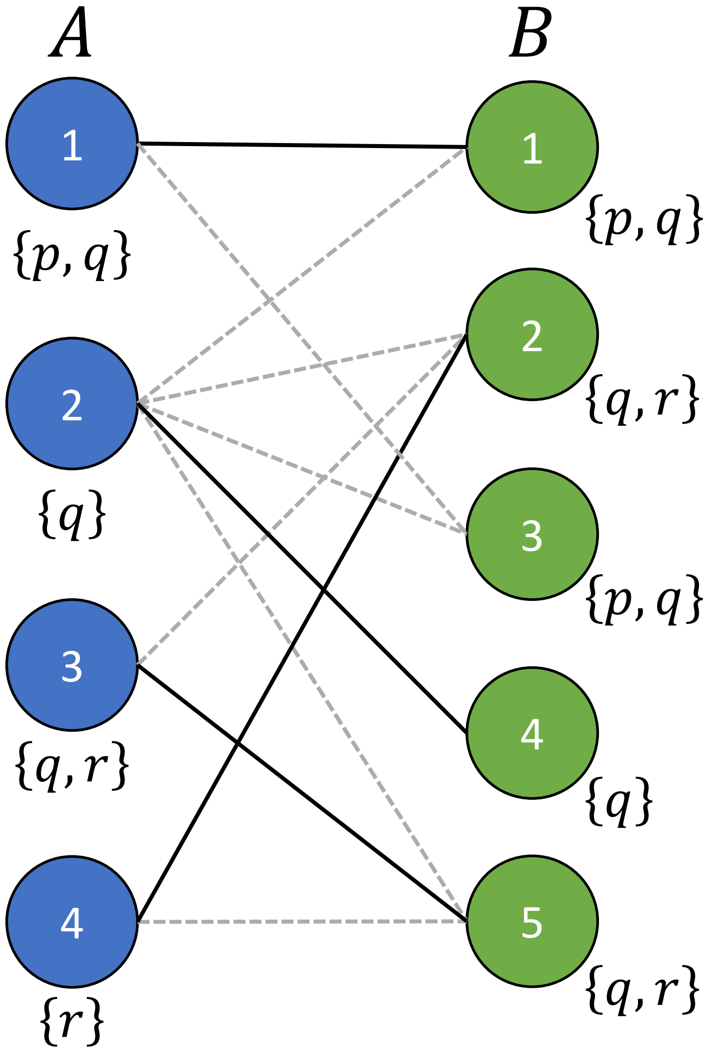


Goals:

- Remove inconsistent edges

Maximum Consistent Sub-Graph

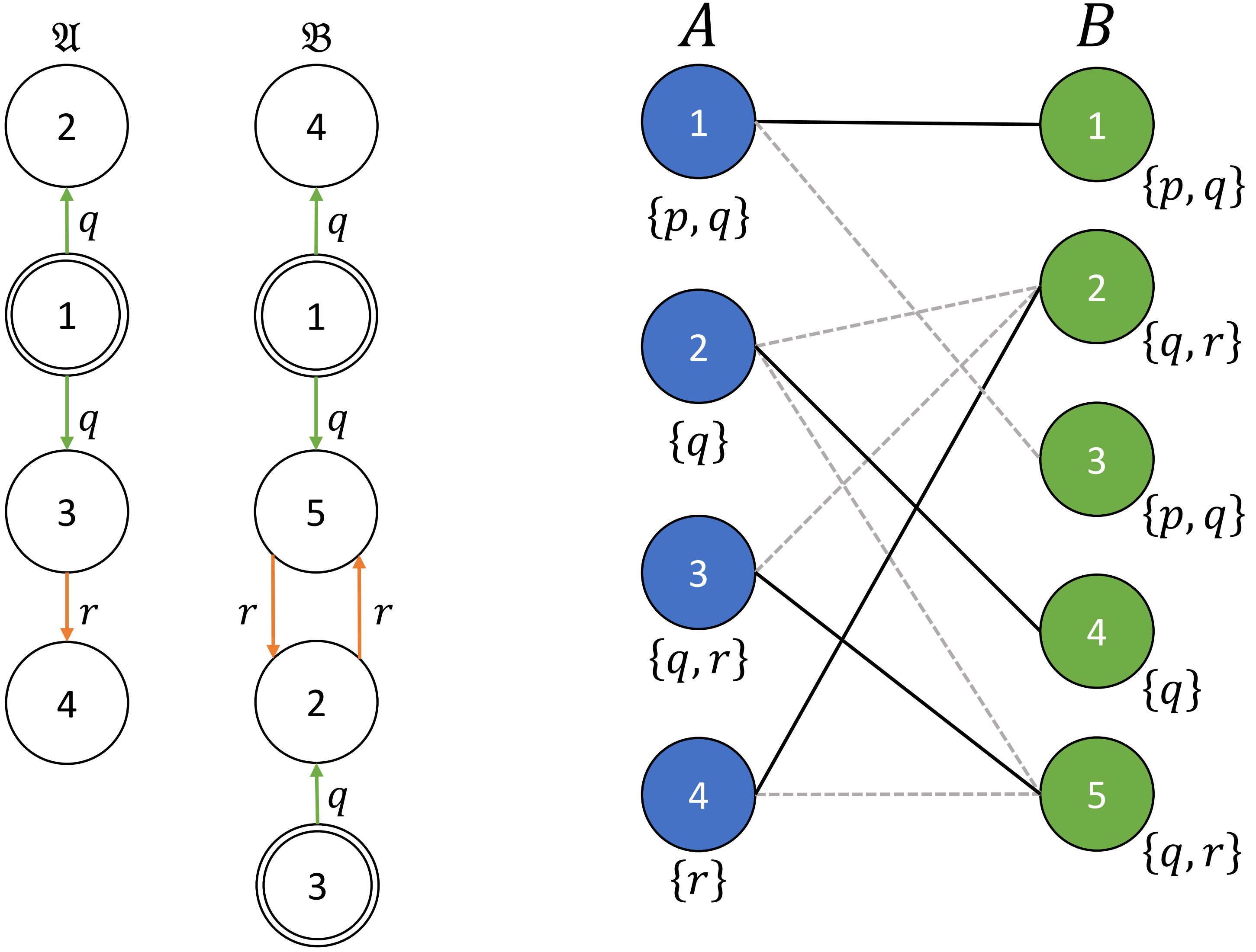




Goals:

- Remove inconsistent edges
- Preserve embeddings

Maximum Consistent Sub-Graph

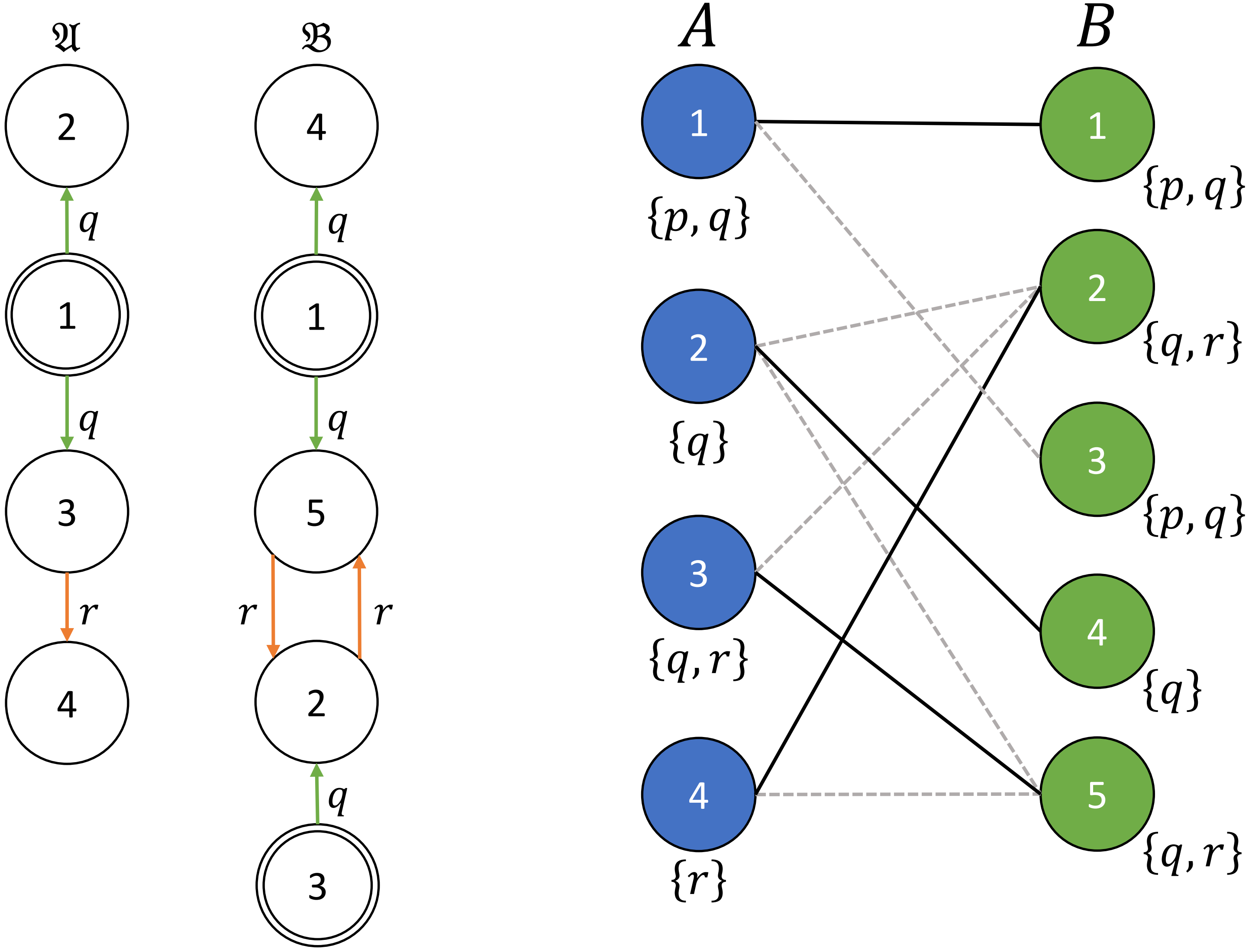


Goals:

- Remove inconsistent edges
- Preserve embeddings
- Efficiently Computable $O(E^2)$
  - Fixpoint Algorithm[1]

[Russel and Norvig. 2009][1]

Maximum Consistent Sub-Graph



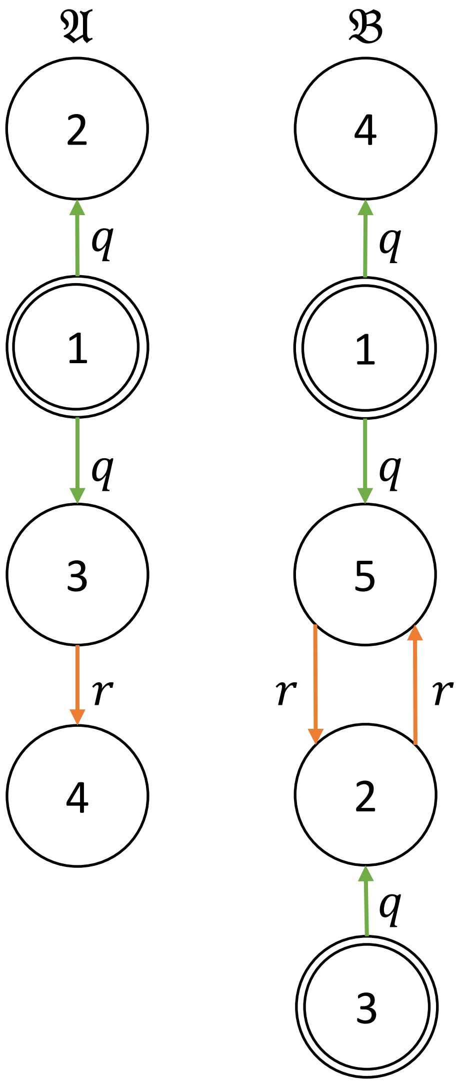
$M_1 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

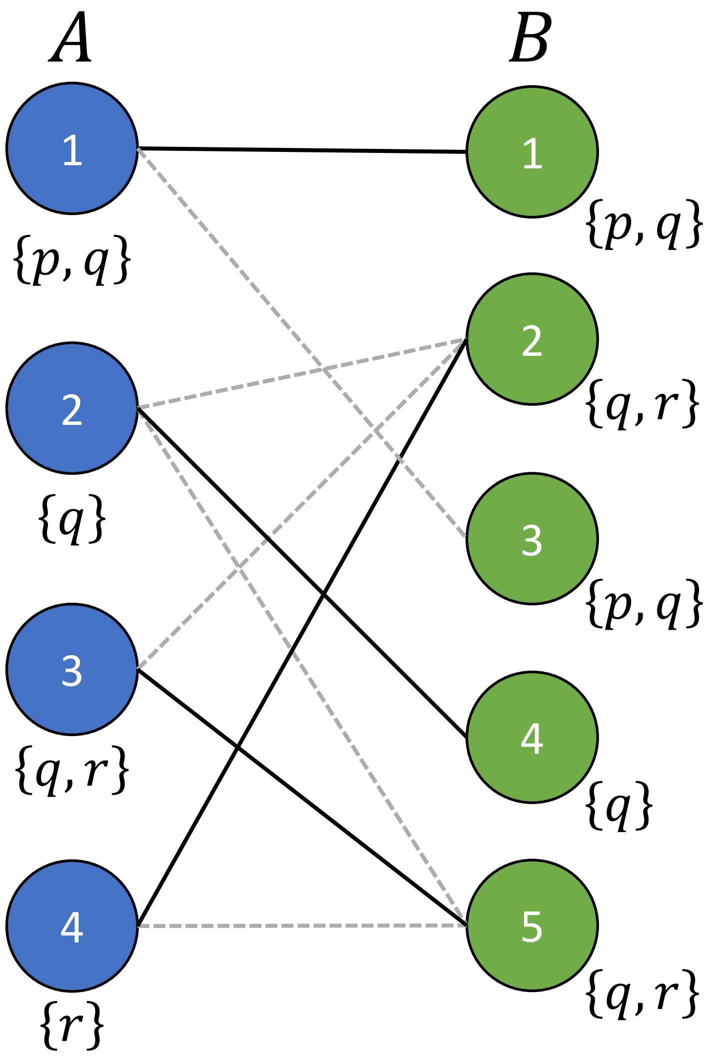
$\boldsymbol{M_2 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}}$

$M_3 \stackrel{\text{def}}{=} \{\langle 1,3\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

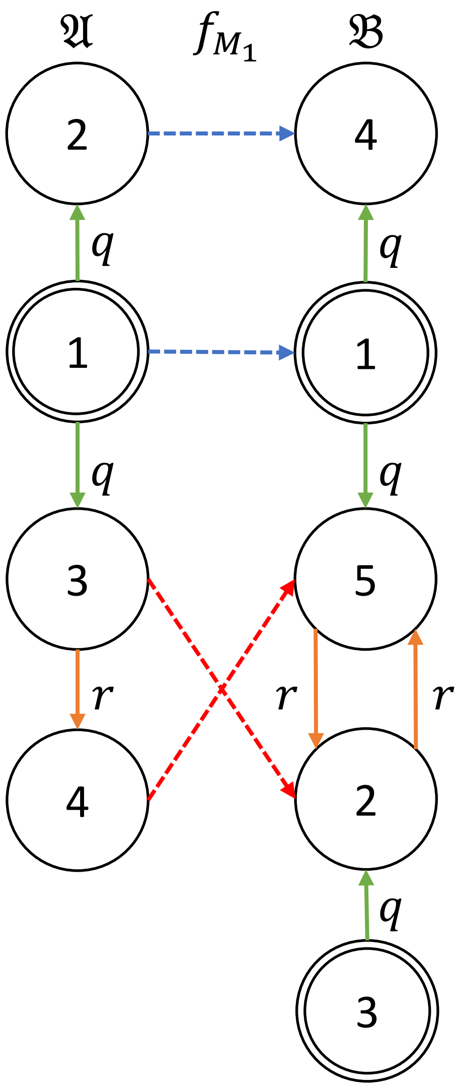
$M_4 \stackrel{\text{def}}{=} \{\langle 1,3\rangle, \langle 2,4\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}$

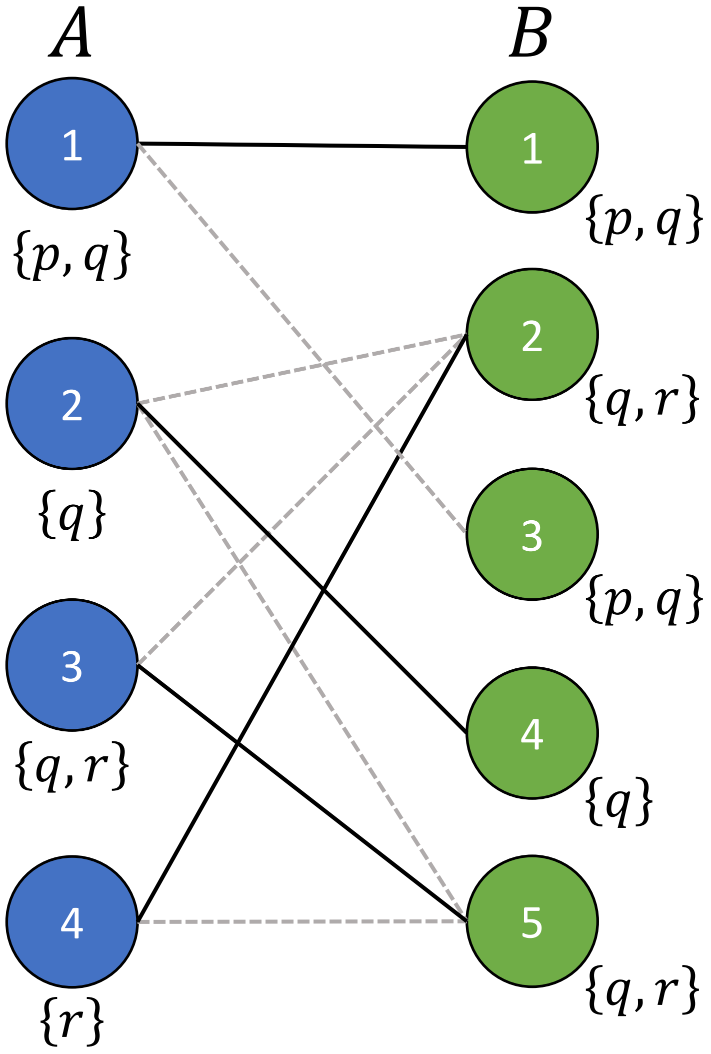
Match Embeds





Match Embeds

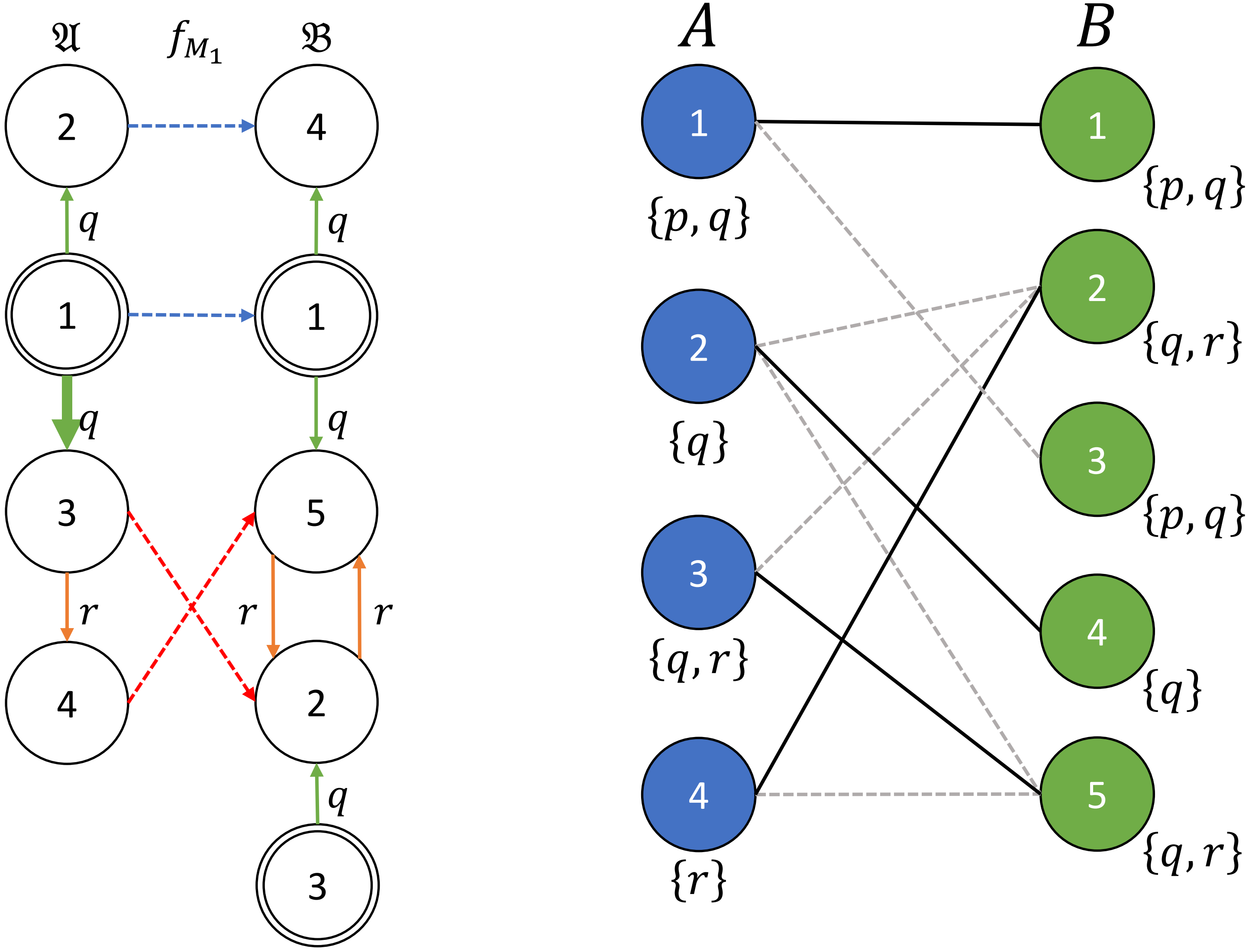




Compute Matching

$M_1 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

Match Embeds

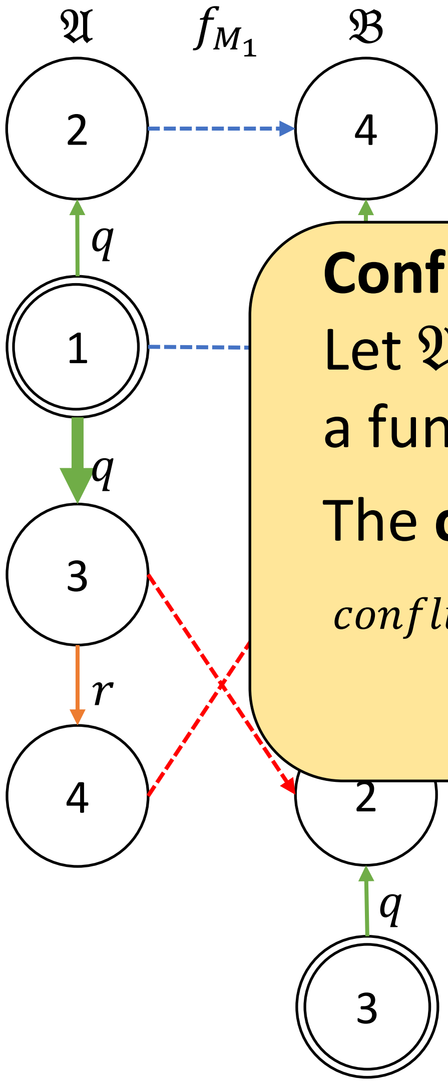


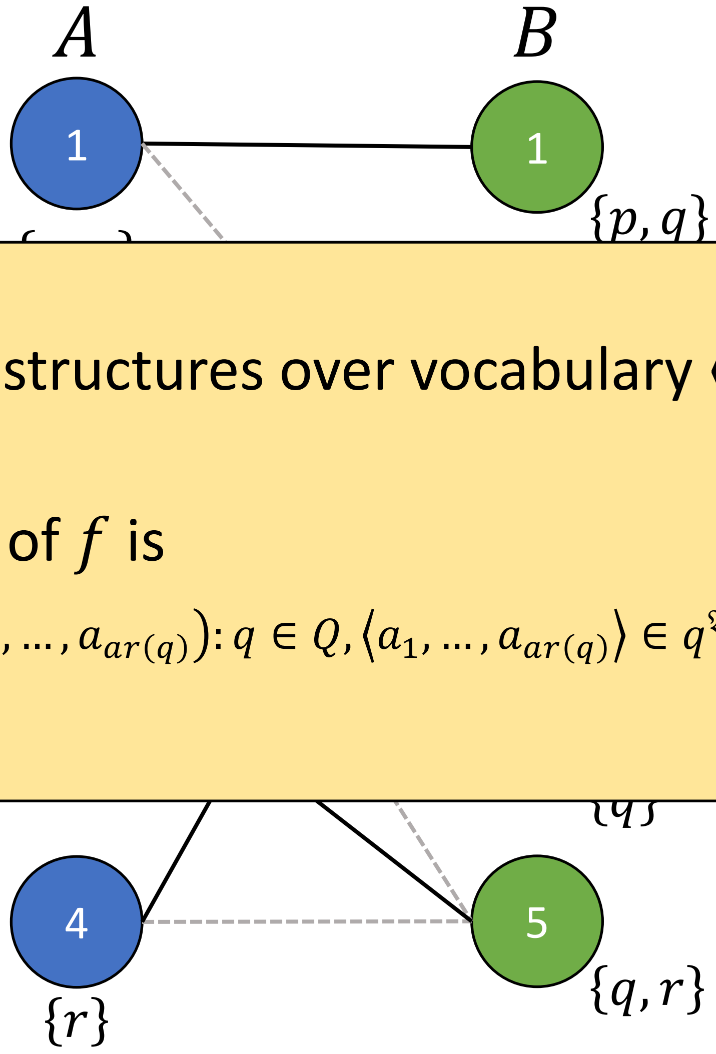
Compute Conflict Set

$$M_1 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$$

$$Conflict(f_{M_1}) \stackrel{\text{def}}{=} \{q(1,3)\}$$

Match Embeds





Compute Conflict Set

$M_1 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

$\{q(1,3)\}$

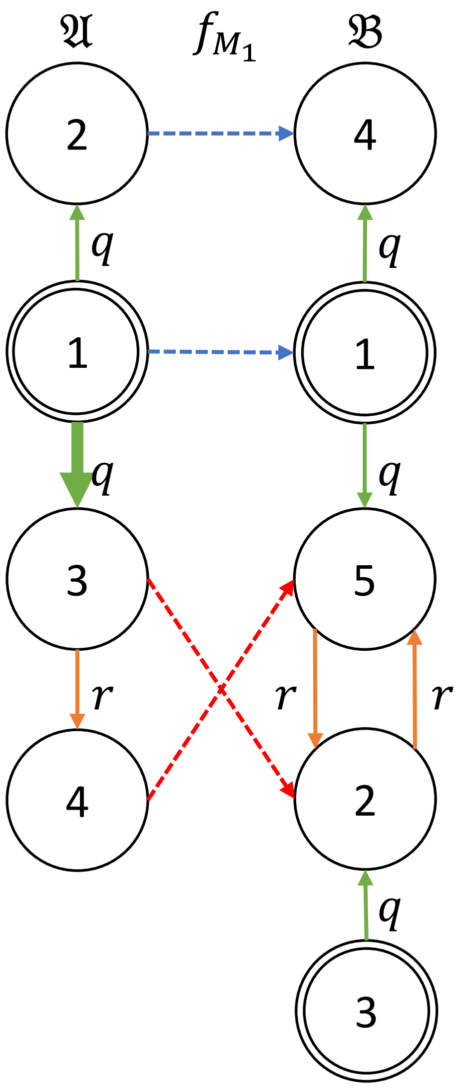
Conflict Set:

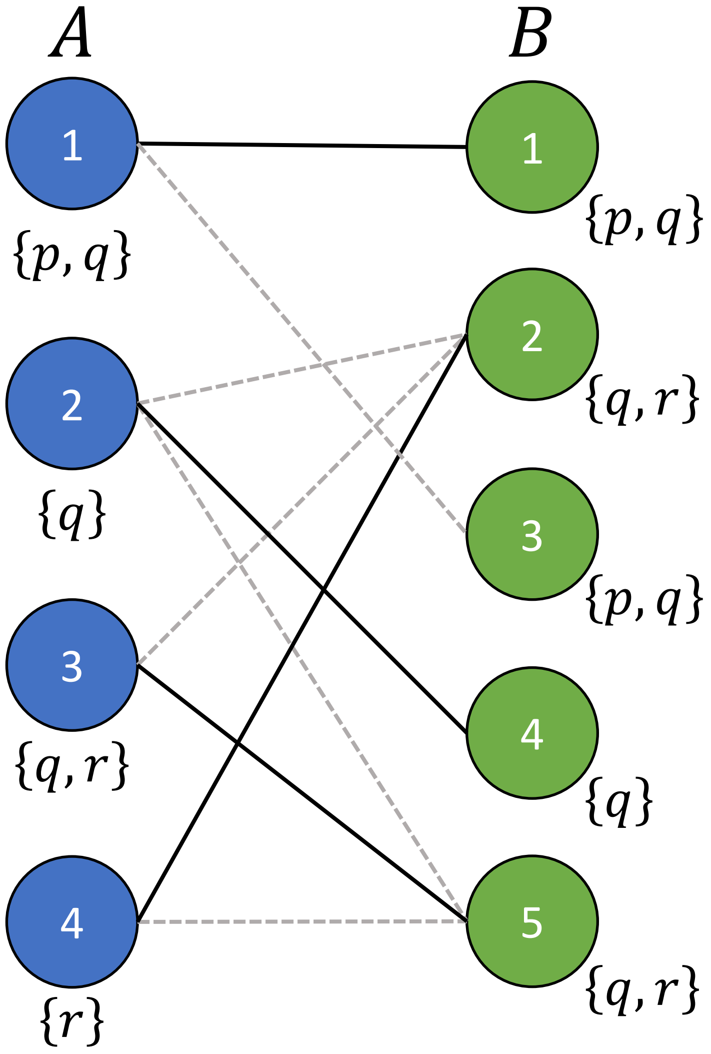
Let $\mathfrak{A}$ and $\mathfrak{B}$ be structures over vocabulary $\langle Q, ar\rangle$ and $f: A \to B$ a function.

The **conflict set** of f is

$$conflict(f) \stackrel{\text{def}}{=} \{q(a_1, \dots, a_{ar(q)}) : q \in Q, \langle a_1, \dots, a_{ar(q)}\rangle \in q^{\mathfrak{A}}, \langle f(a_1), \dots, f(a_{ar(q)})\rangle \in q^{\mathfrak{B}}\}$$

Match Embeds



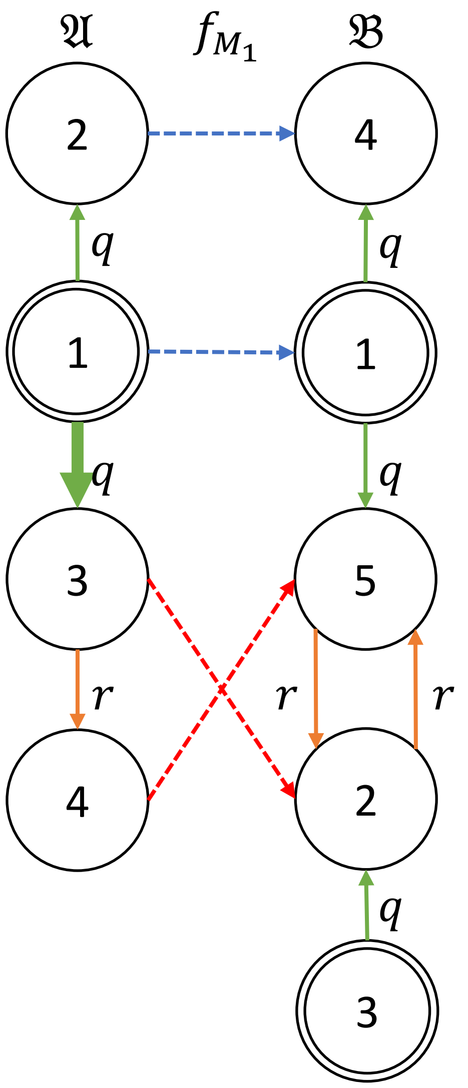


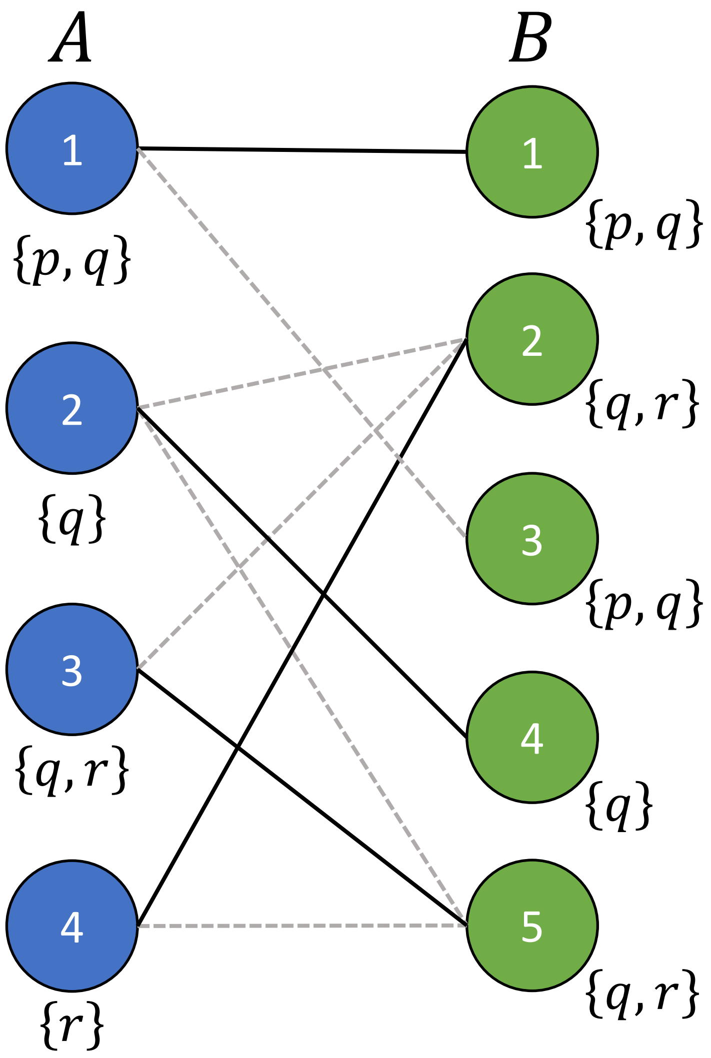
Compute Conflict Set

$M_1 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

$Conflict(f_{M_1}) \stackrel{\text{def}}{=} \{q(1,3)\}$

Match Embeds





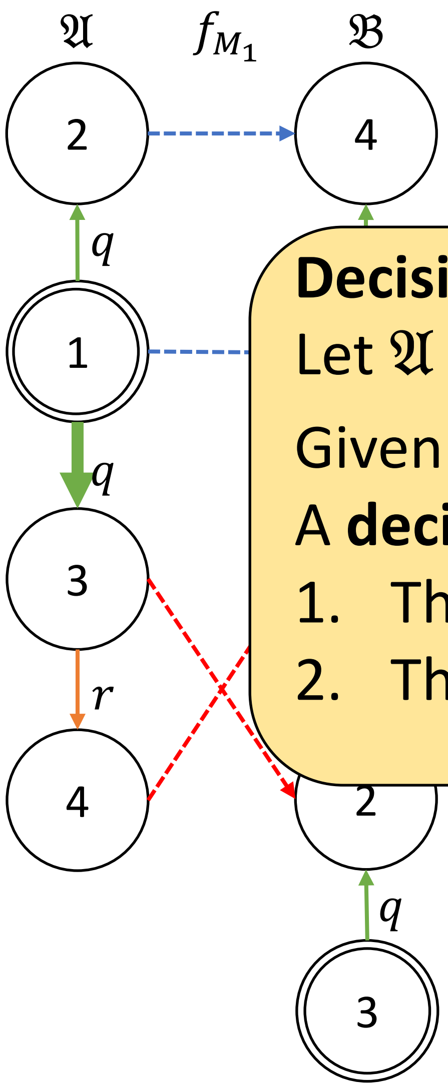
Compute Decisions

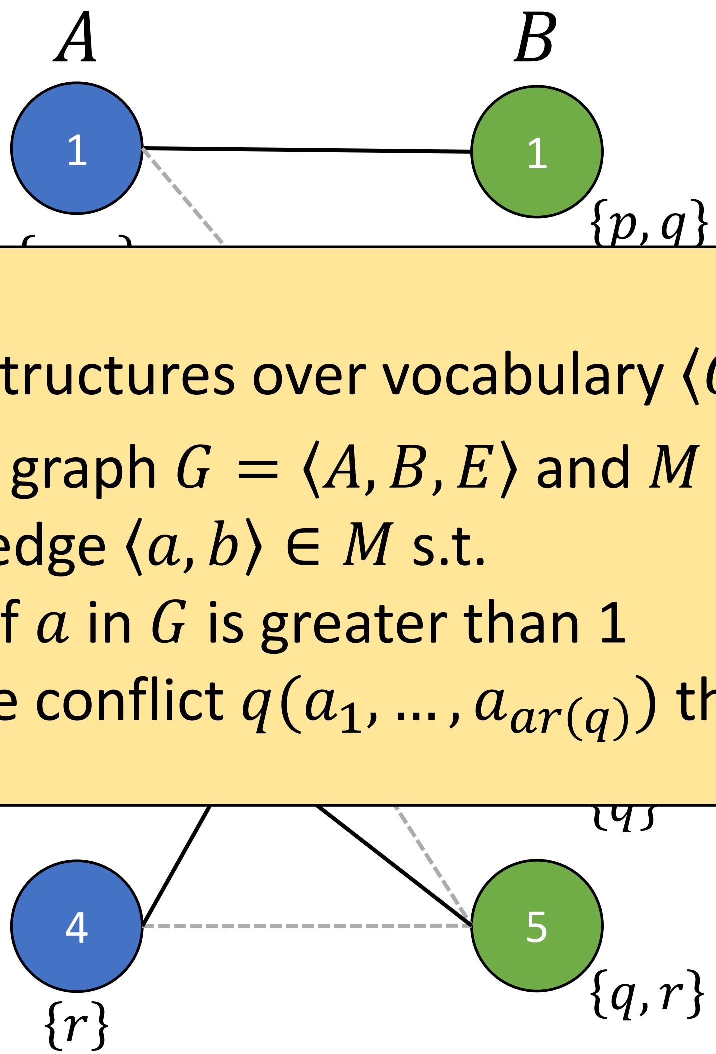
$M_1 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

$Conflict(f_{M_1}) \stackrel{\text{def}}{=} \{q(1,3)\}$

$Decisions(M_1) \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 3,2\rangle\}$

Match Embeds





Compute Decisions

$M_1 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

$\{q(1,3)\}$

$\{\langle 1,1\rangle, \langle 3,2\rangle\}$

Decision:

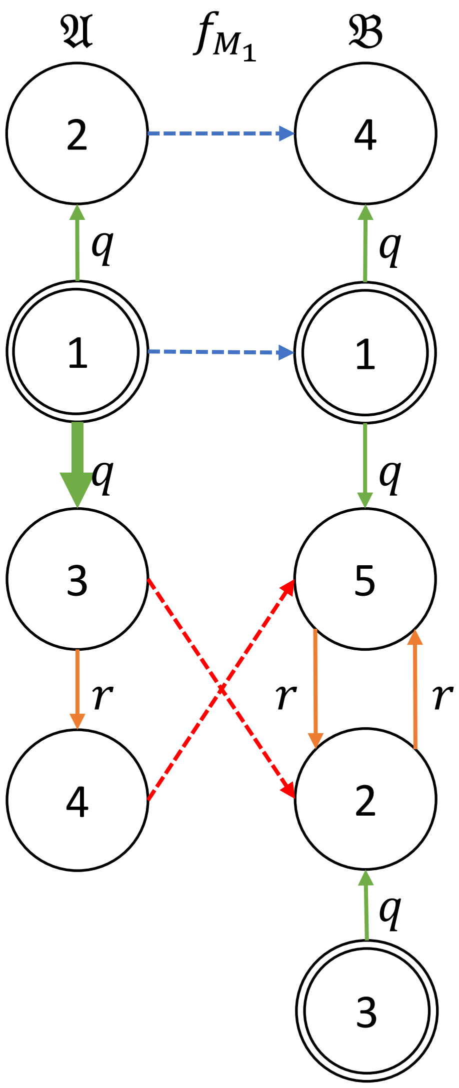
Let 𝔄 and 𝔅 be structures over vocabulary $\langle Q, ar\rangle$.

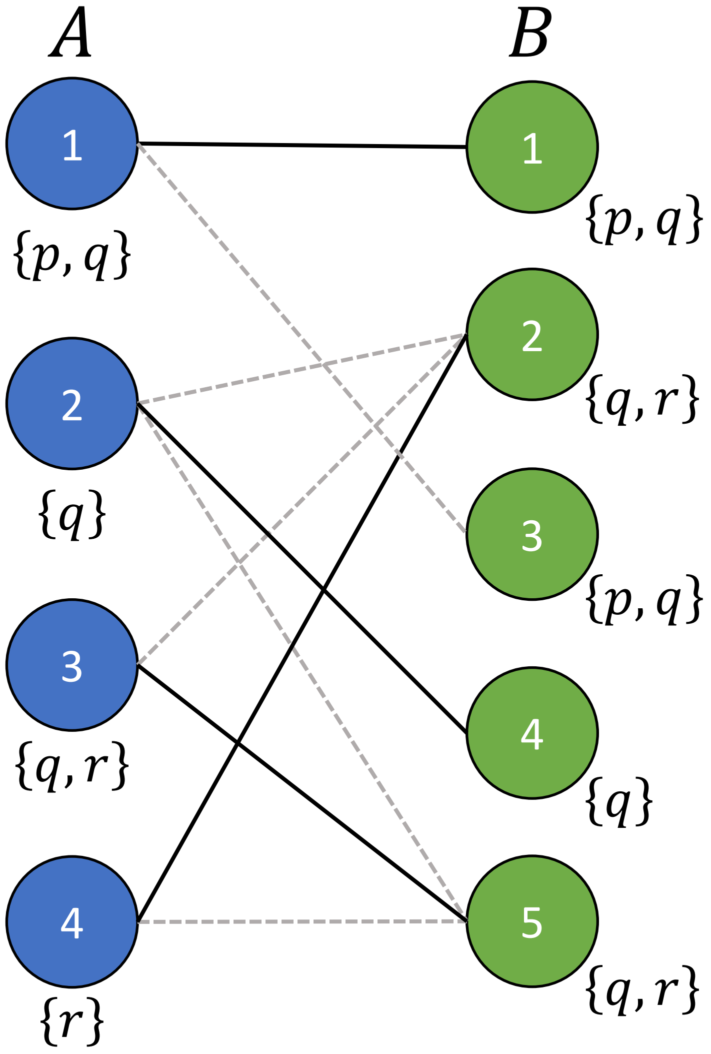
Given a bipartite graph $G = \langle A, B, E\rangle$ and M a matching on G.

A **decision** is an edge $\langle a, b\rangle \in M$ s.t.

1. The degree of a in G is greater than 1
2. There is some conflict $q(a_1, \ldots, a_{ar(q)})$ that involves a ($a = a_i$).

Match Embeds





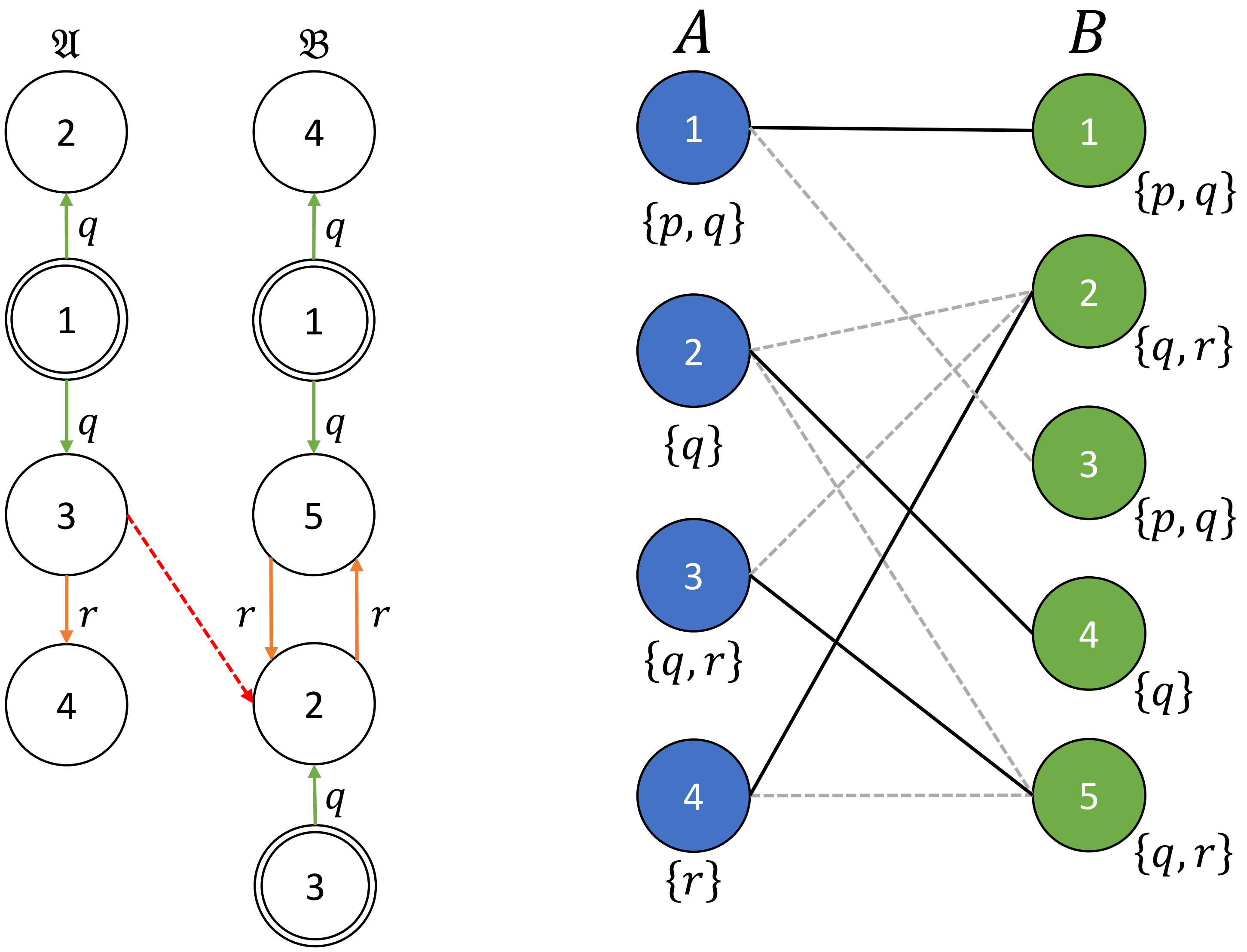
Compute Decisions

$M_1 \overset{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,2\rangle, \langle 4,5\rangle\}$

$Conflict(f_{M_1}) \overset{\text{def}}{=} \{q(1,3)\}$

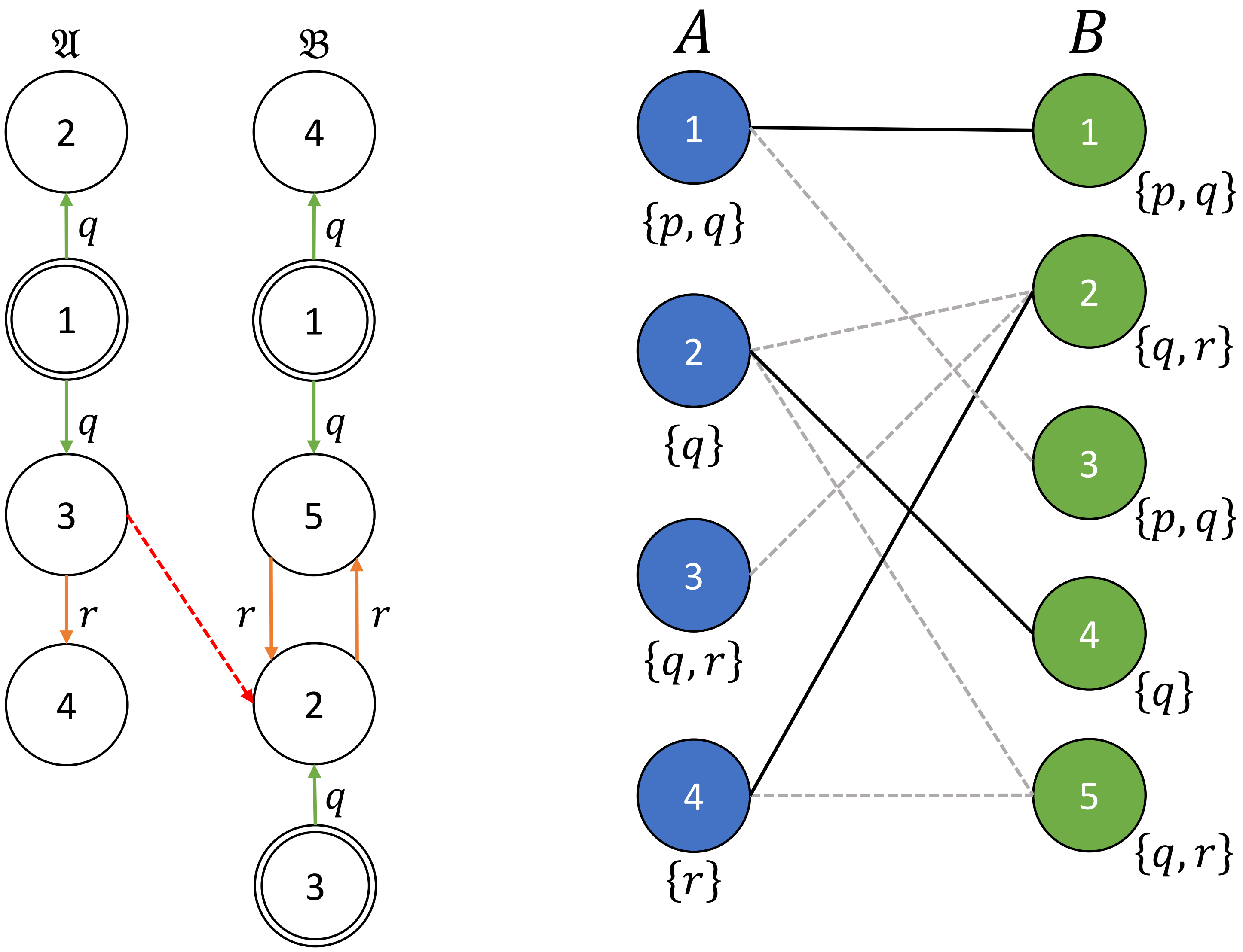
$Decisions(M_1) \overset{\text{def}}{=} \{\langle 1,1\rangle, \langle 3,2\rangle\}$

Match Embeds



Decide $[3 \longmapsto 2]$

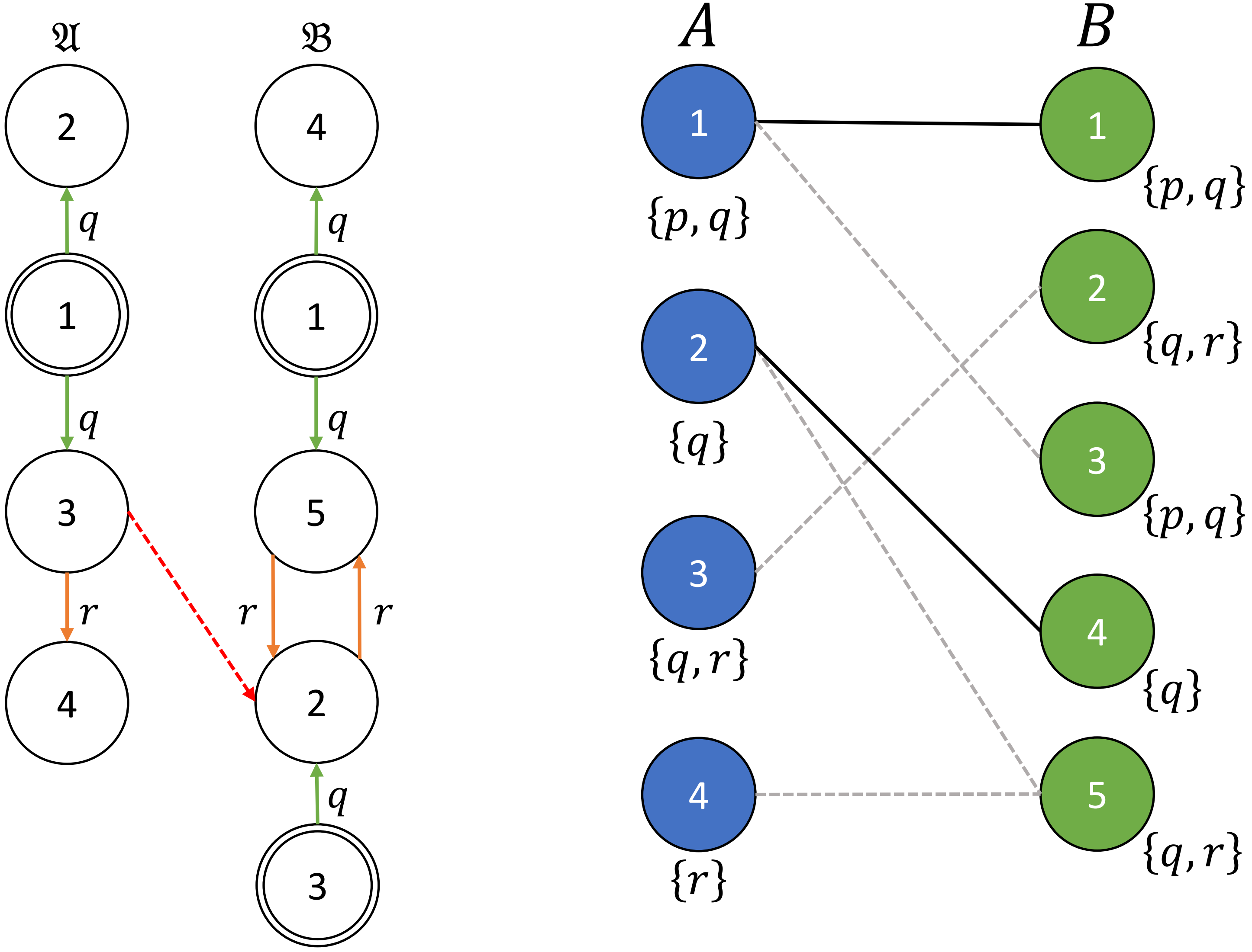
Match Embeds



Decide $[3 \longmapsto 2]$

- Remove ⟨3,5⟩

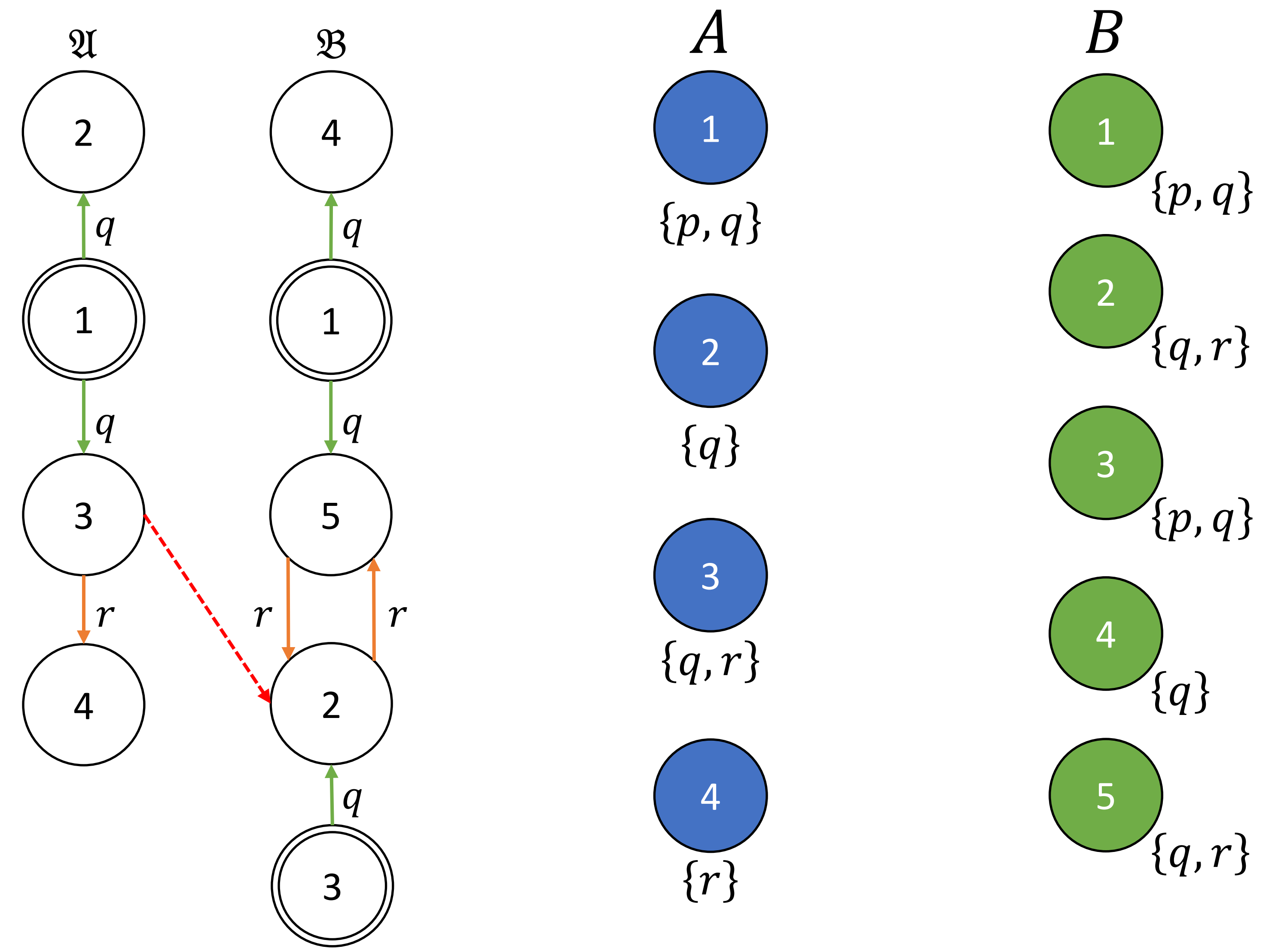
Match Embeds



Decide $[3 \longmapsto 2]$

- Remove $\langle 3,5 \rangle, \langle 2,2 \rangle, \langle 4,2 \rangle$

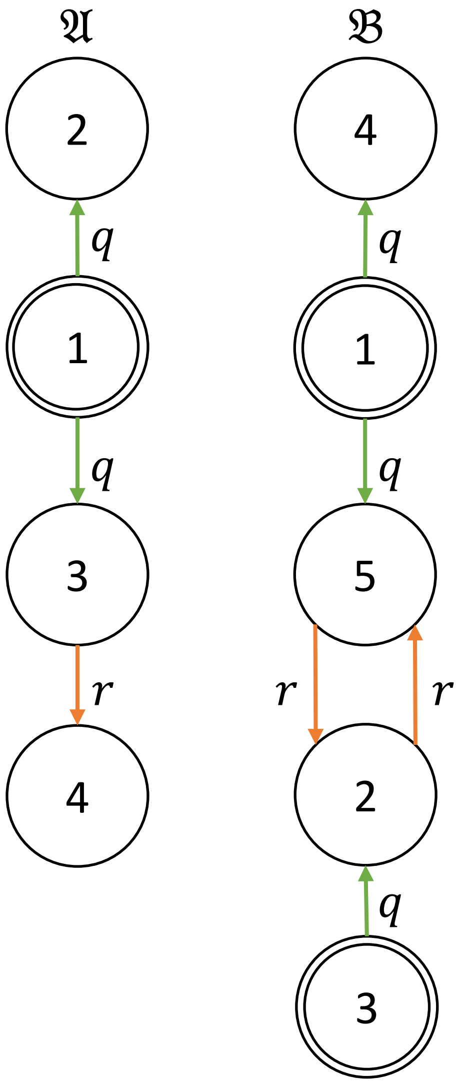
Match Embeds

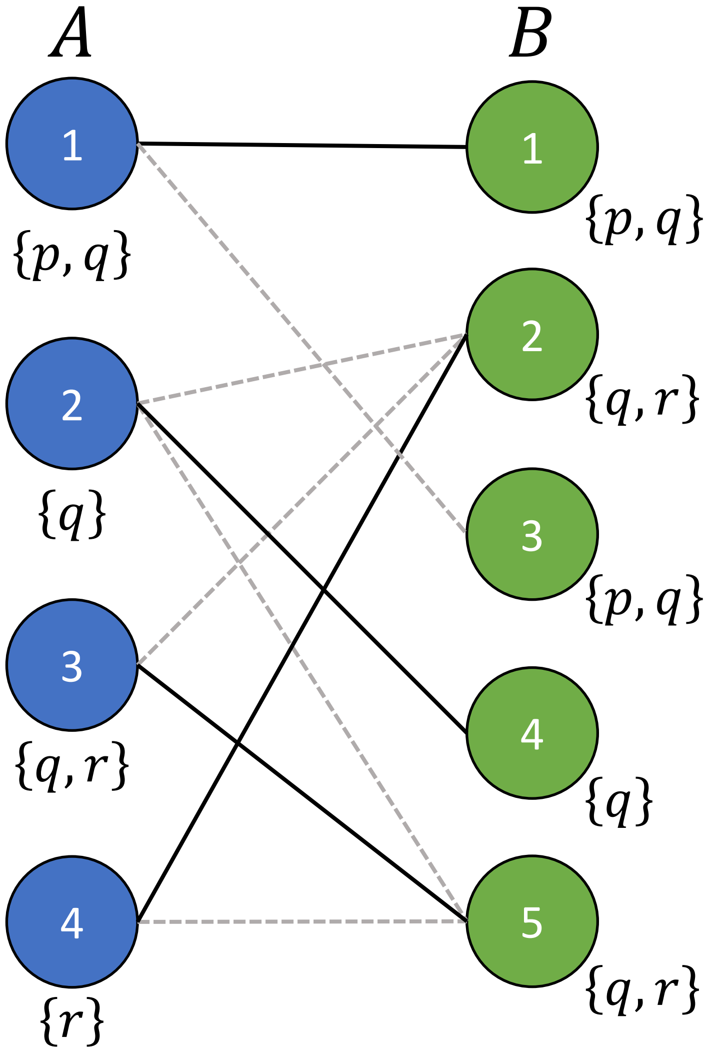


Decide $[3 \longmapsto 2]$

- Remove $\langle 3,5 \rangle, \langle 2,2 \rangle, \langle 4,2 \rangle$
- Compute consistent sub-graph

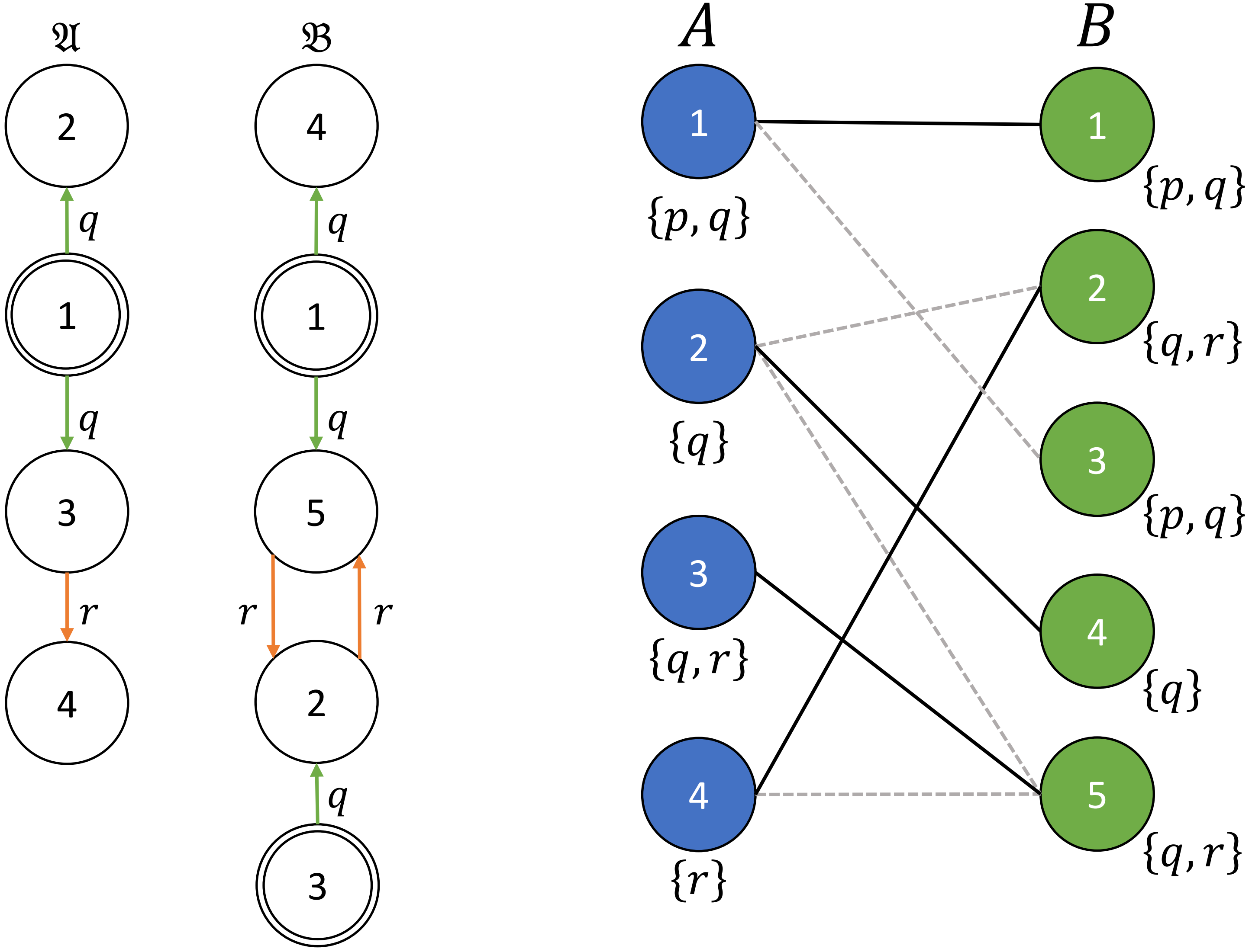
Match Embeds





Backtrack $[3 \longmapsto 2]$

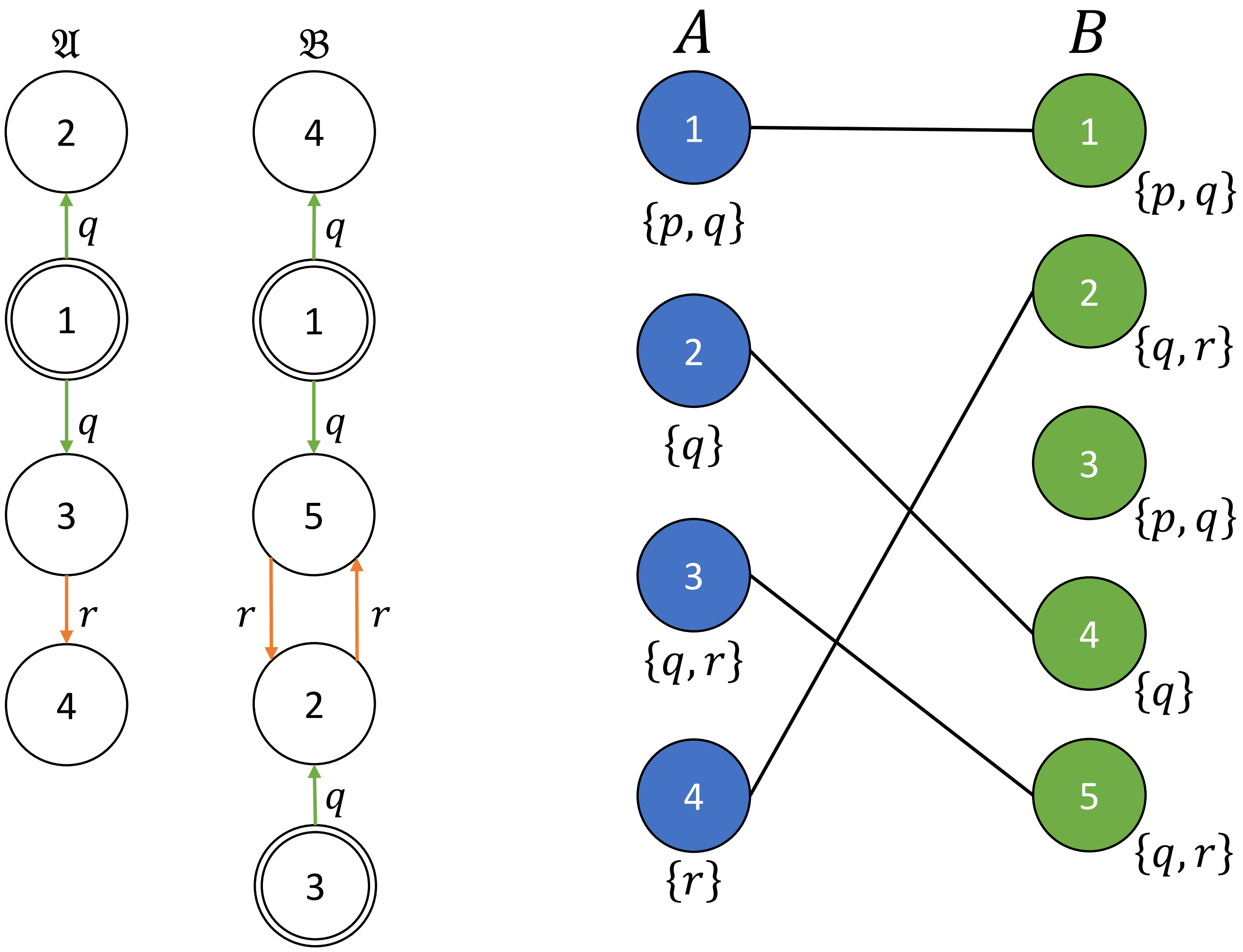
Match Embeds



Backtrack $[3 \longmapsto 2]$

- Blame ⟨3,2⟩

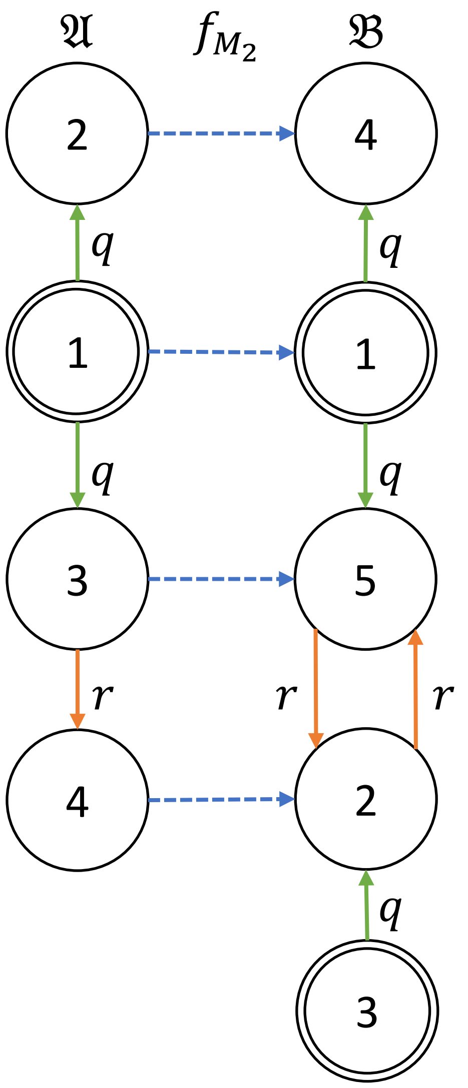
Match Embeds

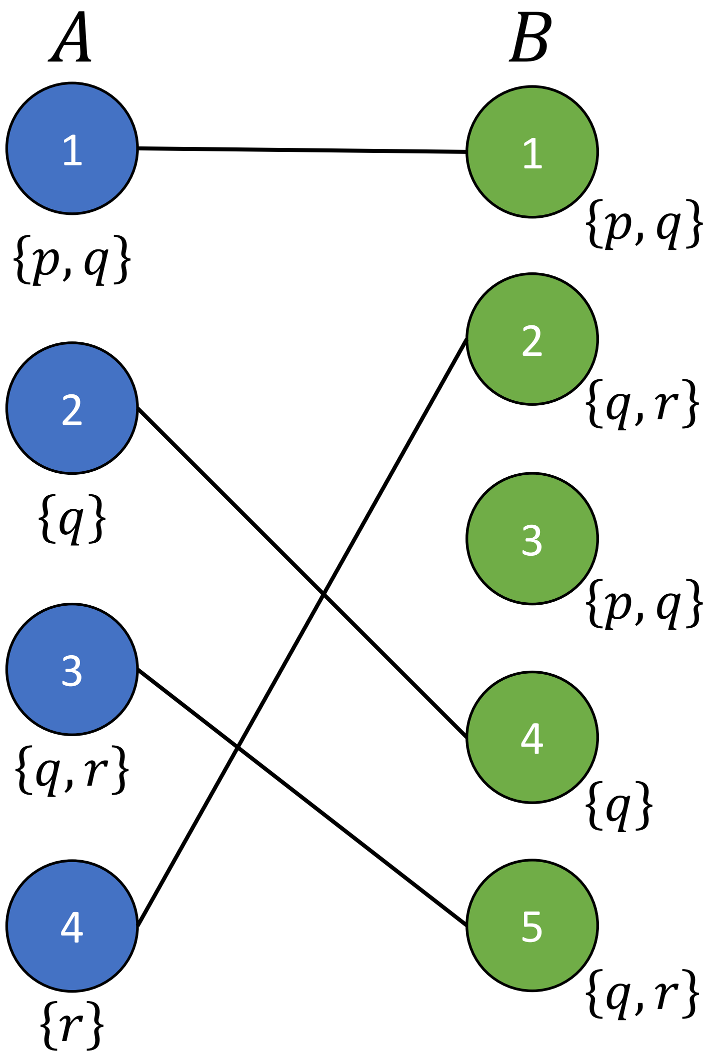


Backtrack $[3 \longmapsto 2]$

- Blame $\langle 3,2 \rangle$
- Compute consistent sub-graph

Match Embeds

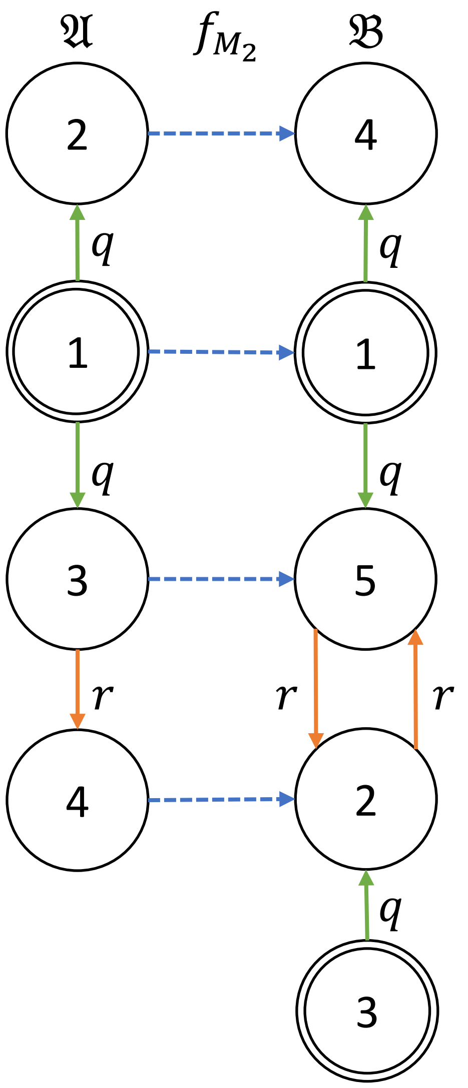


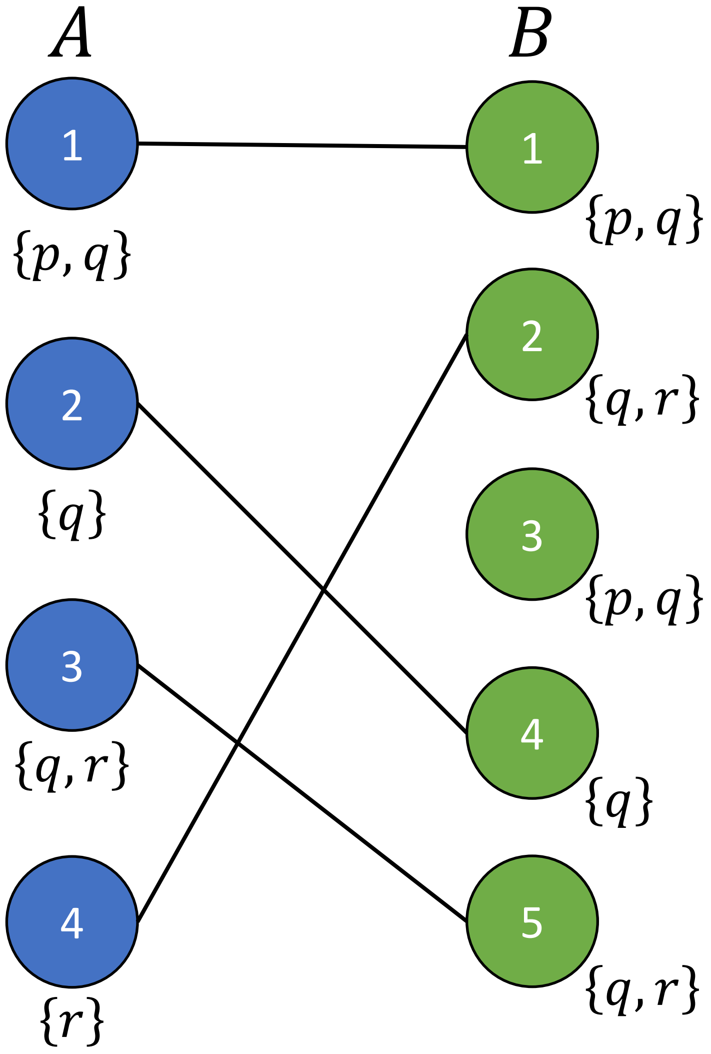


Compute Matching

$M_2 \overset{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}$

Match Embeds



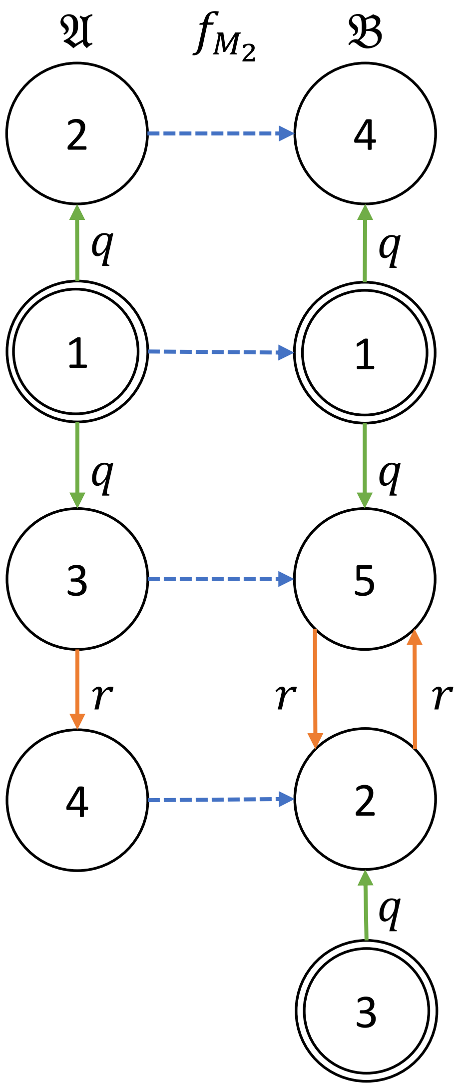


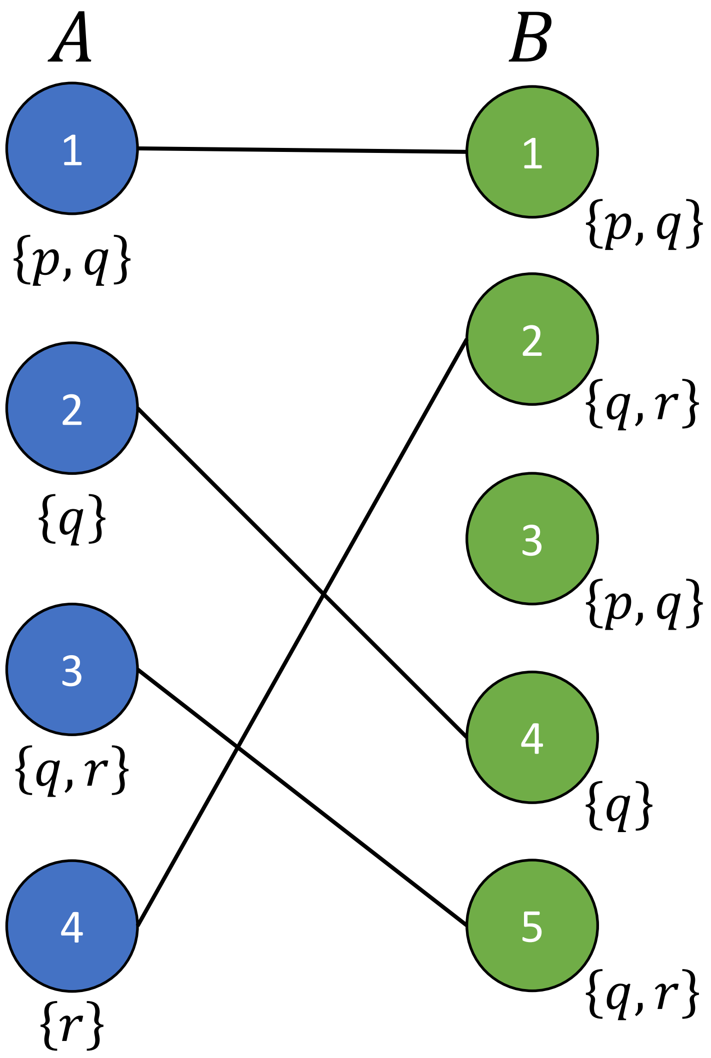
Compute Conflict Set

$M_2 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}$

$Conflict(f_{M_2}) \stackrel{\text{def}}{=} \emptyset$

Match Embeds





Compute Conflict Set

$M_2 \stackrel{\text{def}}{=} \{\langle 1,1\rangle, \langle 2,4\rangle, \langle 3,5\rangle, \langle 4,2\rangle\}$

$Conflict(f_{M_2}) \stackrel{\text{def}}{=} \emptyset$

f_{M_2} is an Embedding

Match Embeds Algorithm

Function embeds(G)
  $G \leftarrow filter(G)$

Match Embeds Algorithm

Function embeds(G)
  $G \leftarrow filter(G)$
  $M \leftarrow$ **maximum_matching**(G)

Match Embeds Algorithm

Function embeds(G)
  $G \leftarrow filter(G)$
  $M \leftarrow$ **maximum_matching**(G)
  if $|M| \neq |G.A|$ **then**
    return false
  end

Match Embeds Algorithm

Function embeds(G)
  $G \leftarrow filter(G)$
  $M \leftarrow$ **maximum_matching**(G)
  if $|M| \neq |G.A|$ **then**
    return false
  end
  if f_M *is an embedding* **then**
    return true
  end

Match Embeds Algorithm

Function embeds(G)
  $G \leftarrow filter(G)$
  $M \leftarrow$ **maximum_matching**(G)
  if $|M| \neq |G.A|$ **then**
    return false
  end
  if f_M *is an embedding* **then**
    return true
  end
  $Select\ a\ decision\ \langle a, b\rangle \in M$

Match Embeds Algorithm

Function embeds(G)
  $G \leftarrow filter(G)$
  $M \leftarrow$ **maximum_matching**(G)
  if $|M| \neq |G.A|$ **then**
    return false
  end
  if f_M *is an embedding* **then**
    return true
  end
  Select a decision $\langle a, b \rangle \in M$
  if embeds$(G \setminus \{\langle u, v \rangle \in E : u = a$ **xor** $v = b\})$ **then**
    return true

Match Embeds Algorithm

Function embeds(G)
  $G \leftarrow filter(G)$
  $M \leftarrow$ **maximum_matching**(G)
  if $|M| \neq |G.A|$ **then**
    return false
  end
  if f_M *is an embedding* **then**
    return true
  end
  Select a decision $\langle a, b \rangle \in M$
  if embeds$(G \backslash \{\langle u, v \rangle \in E : u = a$ **xor** $v = b\})$ **then**
    return true
  else
    return embeds$(G \backslash \{\langle a, b \rangle\})$
  end

Match Embeds

- Inspired by monadic reduction to bipartite graph matching

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings
  1. Remove inconsistent edges from graph

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings
  1. Remove inconsistent edges from graph
  2. Compute maximum matching

Match Embeds

- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings
  1. Remove inconsistent edges from graph
  2. Compute maximum matching
  3. Check for conflicts

Match Embeds

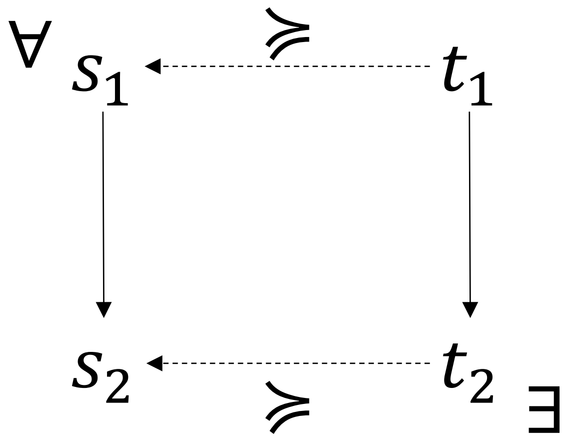
- Inspired by monadic reduction to bipartite graph matching
  - If f_M is a structure embedding then $M \subseteq E$ is a matching covering A
- Backtracking search algorithm over total matchings
  1. Remove inconsistent edges from graph
  2. Compute maximum matching
  3. Check for conflicts
  4. Decide on edges in matching and recurse

Match Embeds for Program verification

- Practical procedure for deciding structure embedding problem

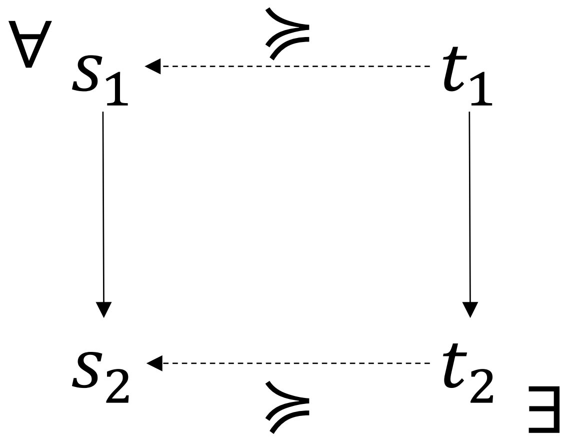
Match Embeds for Program verification

- Practical procedure for deciding structure embedding problem
- For Predicate Automata prune unnecessary branches:



Match Embeds for Program verification

- Practical procedure for deciding structure embedding problem
- For Predicate Automata prune unnecessary branches:



- Need to search for some already explored t_1 to prune s_1.

Multi-Source Single-Target Embeddings

- Check if $\mathfrak{B}$ embeds a structure within a set of structures
  - $\exists \mathfrak{A} \in Str.\ \mathfrak{A}$ embeds into $\mathfrak{B}$

Multi-Source Single-Target Embeddings

- Check if $\mathfrak{B}$ embeds a structure within a set of structures
  - $\exists \mathfrak{A} \in Str.\ \mathfrak{A}$ embeds into $\mathfrak{B}$
- Key idea: no need to check all structures

Multi-Source Single-Target Embeddings

- Check if $\mathfrak{B}$ embeds a structure within a set of structures
  - $\exists \mathfrak{A} \in Str.\ \mathfrak{A}$ embeds into $\mathfrak{B}$
- Key idea: no need to check all structures
  - Store structures in a k-d tree

Multi-Source Single-Target Embeddings

- Check if $\mathfrak{B}$ embeds a structure within a set of structures
  - $\exists \mathfrak{A} \in Str.\ \mathfrak{A}$ embeds into $\mathfrak{B}$
- Key idea: no need to check all structures
  - Store structures in a k-d tree
  - Map each $\mathfrak{A}$ to $v(\mathfrak{A}) \in \mathbb{N}^d$

Multi-Source Single-Target Embeddings

- Check if $\mathfrak{B}$ embeds a structure within a set of structures
  - $\exists \mathfrak{A} \in Str.\ \mathfrak{A}$ embeds into $\mathfrak{B}$
- Key idea: no need to check all structures
  - Store structures in a k-d tree
  - Map each $\mathfrak{A}$ to $v(\mathfrak{A}) \in \mathbb{N}^d$
  - If $\mathfrak{A}$ embeds into $\mathfrak{B}$ then $v(\mathfrak{A}) \leq v(\mathfrak{B})$

Multi-Source Single-Target Embeddings

- Check if $\mathfrak{B}$ embeds a structure within a set of structures
  - $\exists \mathfrak{A} \in Str.\ \mathfrak{A}$ embeds into $\mathfrak{B}$
- Key idea: no need to check all structures
  - Store structures in a k-d tree
  - Map each $\mathfrak{A}$ to $v(\mathfrak{A}) \in \mathbb{N}^d$
  - If $\mathfrak{A}$ embeds into $\mathfrak{B}$ then $v(\mathfrak{A}) \leq v(\mathfrak{B})$
  - Use range queries on k-d tree and test returned structures

Multi-Source Single-Target Embeddings

- Let structures be over vocabulary $\langle Q, ar \rangle$
  - v maps structures to $2^{|Q|}$ vectors
  - $v(\mathfrak{A})_i = 1 \Leftrightarrow q_i^{\mathfrak{A}} \neq \emptyset \quad (q_i(\dots) \in \mathfrak{A})$

Multi-Source Single-Target Embeddings

- Let structures be over vocabulary $\langle Q, ar \rangle$
  - v maps structures to $2^{|Q|}$ vectors
  - $v(\mathfrak{A})_i = 1 \Leftrightarrow q_i^{\mathfrak{A}} \neq \emptyset \quad (q_i(\dots) \in \mathfrak{A})$

If $\mathfrak{A}$ embeds into $\mathfrak{B}$

$$v(\mathfrak{A})_i = 1 \implies v(\mathfrak{B})_i = 1$$

$$v(\mathfrak{A}) \leq v(\mathfrak{B})$$

Multi-Source Single-Target Embeddings

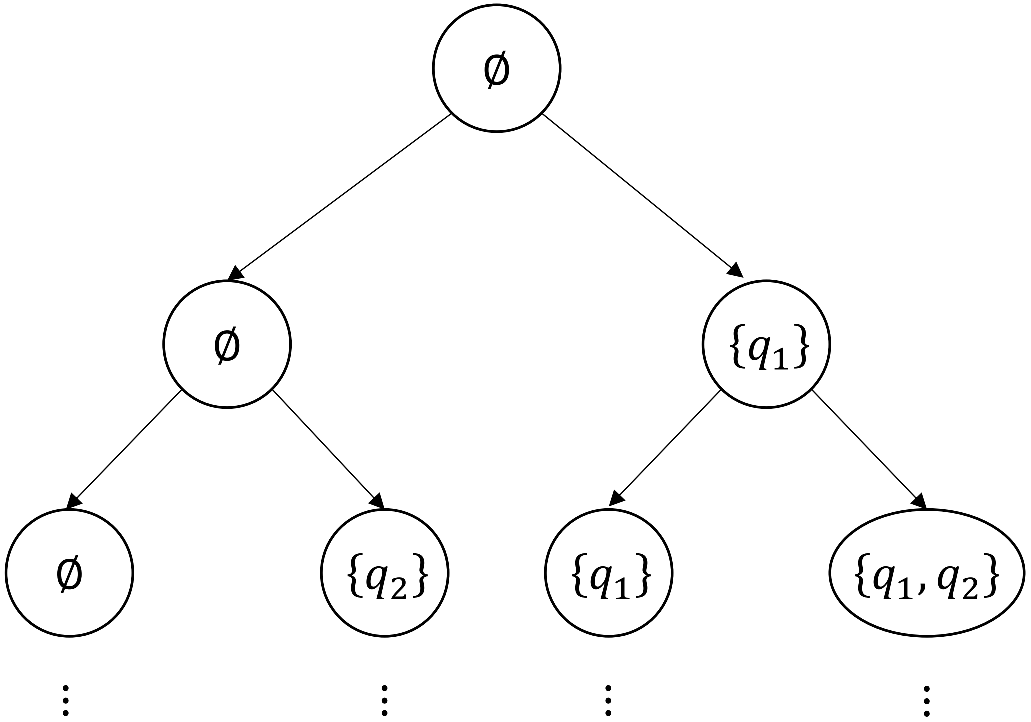
- Let structures be over vocabulary $\langle Q, ar \rangle$
  - v maps structures to $2^{|Q|}$ vectors
  - $v(\mathfrak{A})_i = 1 \Leftrightarrow q_i^{\mathfrak{A}} \neq \emptyset \quad (q_i(\ldots) \in \mathfrak{A})$

If $\mathfrak{A}$ embeds into $\mathfrak{B}$

$$v(\mathfrak{A})_i = 1 \implies v(\mathfrak{B})_i = 1$$

$$v(\mathfrak{A}) \leq v(\mathfrak{B})$$

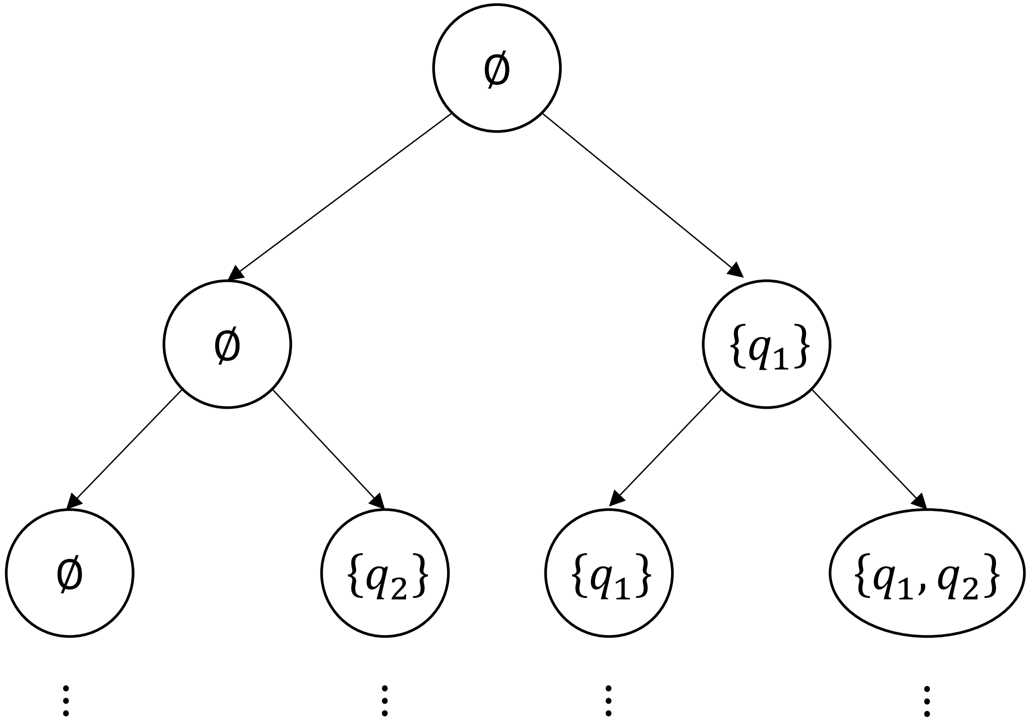
k-d Tree Structure



Multi-Source Single-Target Embeddings

- Range Query: $\mathfrak{A} = \langle q_2(1), q_2(2) \rangle$
  1. Check root
  2. Check left tree
  3. At level i check right tree if $q_{i+1}^{\mathfrak{A}} \neq \emptyset$

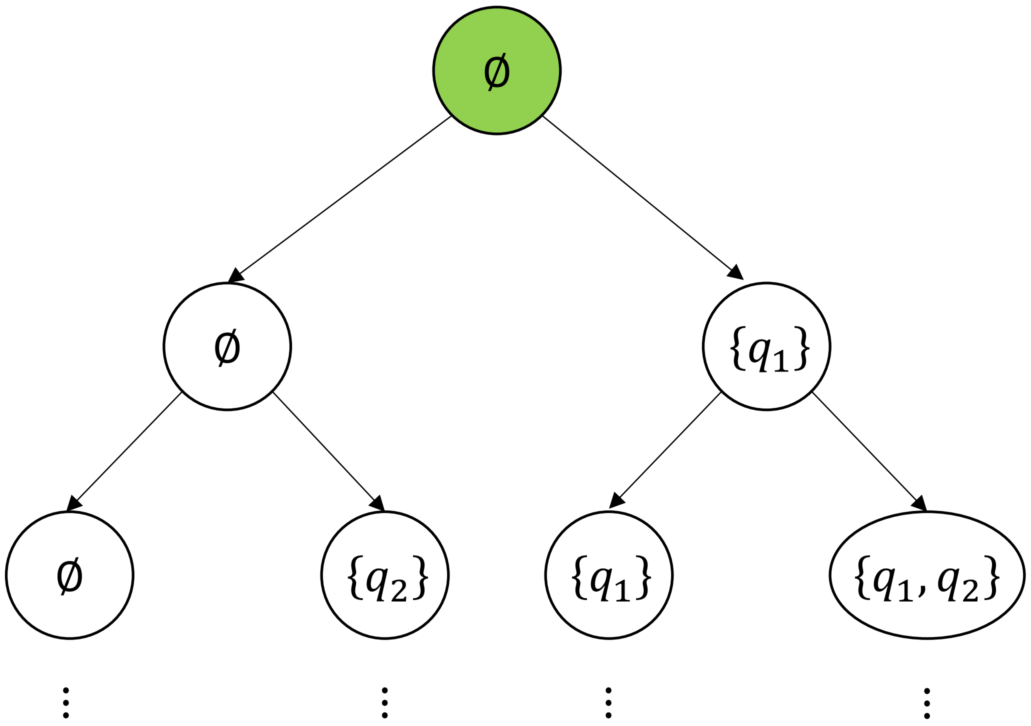
k-d Tree Structure



Multi-Source Single-Target Embeddings

- Range Query: $\mathfrak{A} = \langle q_2(1), q_2(2) \rangle$
  1. Check root
  2. Check left tree
  3. At level i check right tree if $q_{i+1}^{\mathfrak{A}} \neq \emptyset$

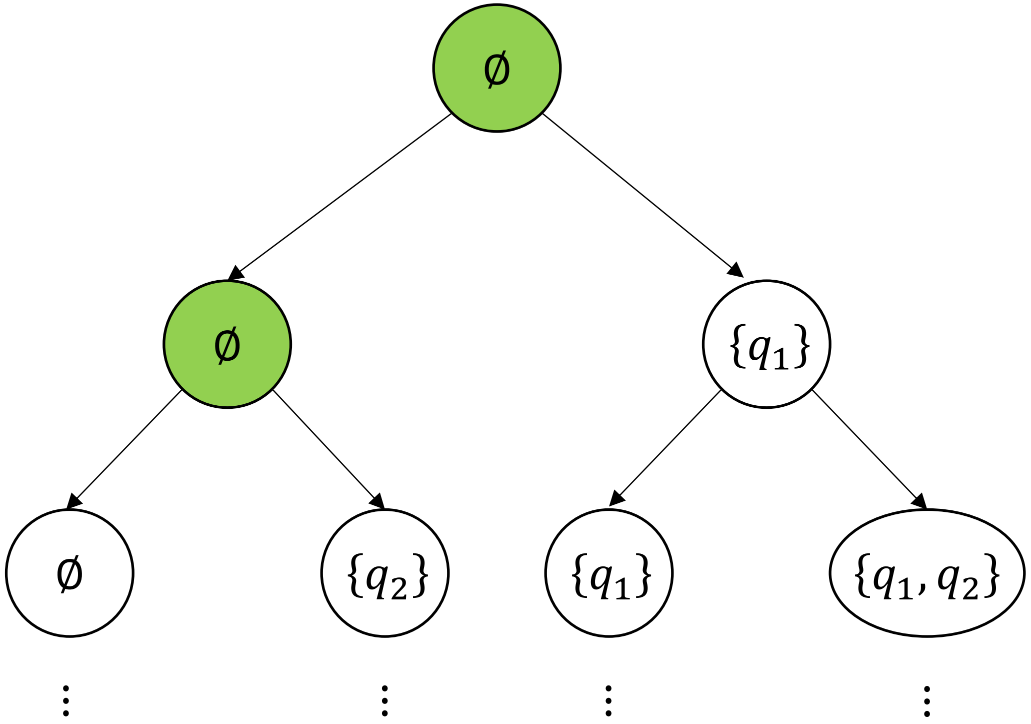
$\boldsymbol{k}$-$\boldsymbol{d}$ Tree Structure



Multi-Source Single-Target Embeddings

- Range Query: $\mathfrak{A} = \langle q_2(1), q_2(2) \rangle$
  1. Check root
  2. Check left tree
  3. At level i check right tree if $q_{i+1}^{\mathfrak{A}} \neq \emptyset$

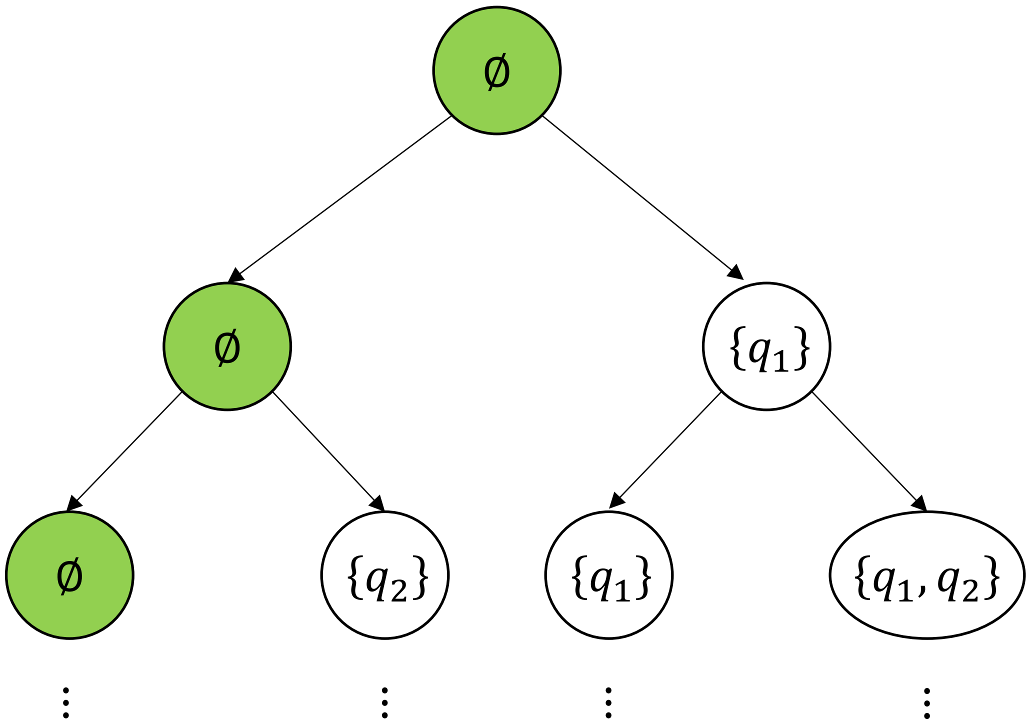
k-d Tree Structure



Multi-Source Single-Target Embeddings

- Range Query: $\mathfrak{A} = \langle q_2(1), q_2(2) \rangle$
  1. Check root
  2. Check left tree
  3. At level i check right tree if $q_{i+1}^{\mathfrak{A}} \neq \emptyset$

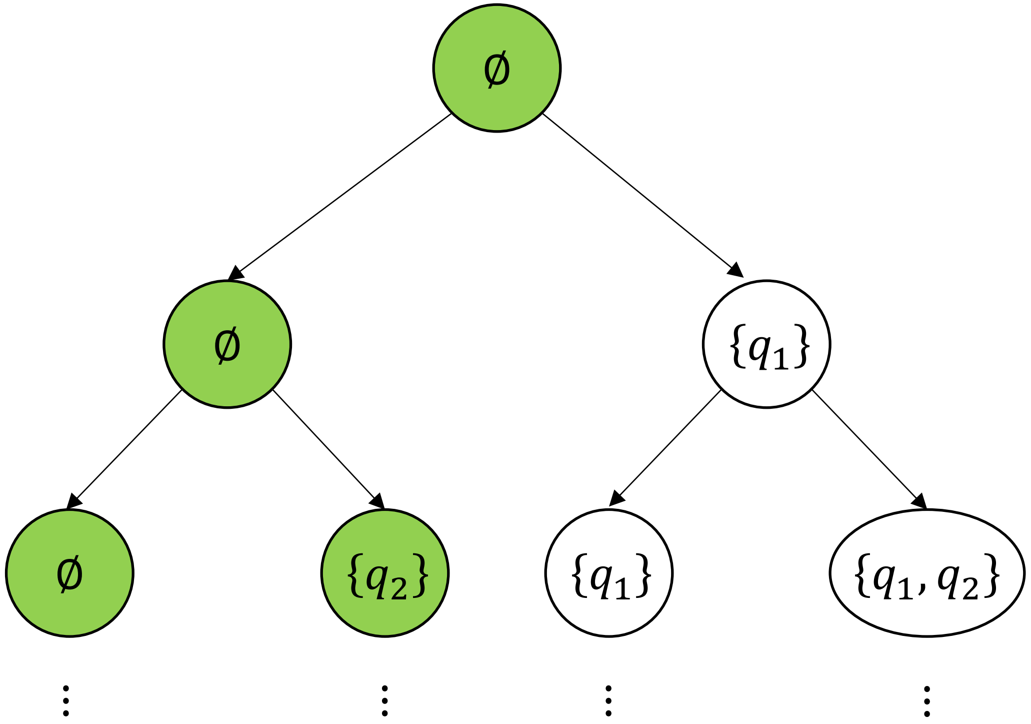
k-d Tree Structure



Multi-Source Single-Target Embeddings

- Range Query: $\mathfrak{A} = \langle q_2(1), q_2(2) \rangle$
  1. Check root
  2. Check left tree
  3. At level i check right tree if $q_{i+1}^{\mathfrak{A}} \neq \emptyset$

k-d Tree Structure



Experiments

- Is Match embeds Practical?

Experiments

- Is Match embeds Practical?
  - Does it improve performance of Proof Spaces?

Experiments

- Is Match embeds Practical?
  - Does it improve performance of Proof Spaces?
  - Does the k-d structure improve Proof Spaces?

Experiments

- Is Match embeds Practical?
  - Does it improve performance of Proof Spaces?
  - Does the k-d structure improve Proof Spaces?

- Compared to Constraint Satisfaction Problem Solvers:
  - Gecode - a top competitor in MiniZinc (CSP Competition)
  - HaifaCSP - 1st prize in 2017 MiniZinc competition
  - OrTool's - Google's Optimization/CSP solver

Constraint Satisfaction Problem

Given structures $\mathfrak{A} = \langle A, q_1, \ldots, q_n \rangle$ and $\mathfrak{B} = \langle B, p_1, \ldots, p_m \rangle$

Constraint Satisfaction Problem

Given structures $\mathfrak{A} = \langle A, q_1, \ldots, q_n \rangle$ and $\mathfrak{B} = \langle B, p_1, \ldots, p_m \rangle$
For each $a \in A$:
  create variable X_a with domain $\{b \in B: sig(\mathfrak{A}, a) \subseteq sig(\mathfrak{B}, b)\}$

Constraint Satisfaction Problem

Given structures $\mathfrak{A} = \langle A, q_1, \dots, q_n \rangle$ and $\mathfrak{B} = \langle B, p_1, \dots, p_m \rangle$
For each $a \in A$:
  create variable X_a with domain $\{b \in B : sig(\mathfrak{A}, a) \subseteq sig(\mathfrak{B}, b)\}$

For each $\langle a, a' \rangle \in A \times A \ s.t. a \ \neq a'$: (all-different)
  create constraint $X_a \neq X_{a'}$

Constraint Satisfaction Problem

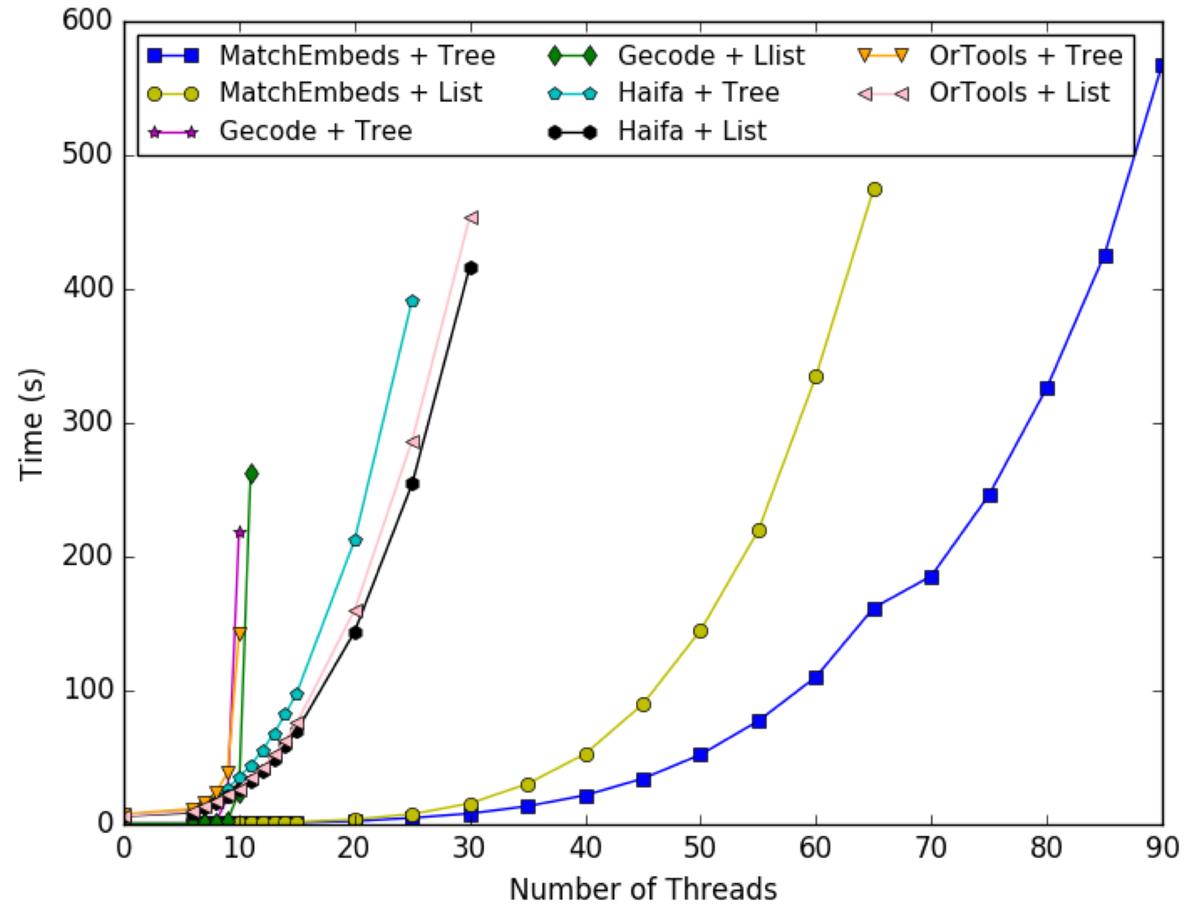
Given structures $\mathfrak{A} = \langle A, q_1, \dots, q_n \rangle$ and $\mathfrak{B} = \langle B, p_1, \dots, p_m \rangle$
For each $a \in A$:
  create variable X_a with domain $\{b \in B: sig(\mathfrak{A}, a) \subseteq sig(\mathfrak{B}, b)\}$

For each $\langle a, a' \rangle \in A \times A\ s.t.\ a \neq a'$: (all-different)
  create constraint $X_a \neq X_{a'}$

For each $q_i \in \mathfrak{A}$ and each $\left\langle a_1, \dots, a_{ar(q_i)} \right\rangle \in q_i^{\mathfrak{A}}$:
  create constraint $\left\langle X_{a_1}, \dots, X_{a_{ar(q_i)}} \right\rangle \in q_i^{\mathfrak{B}}$

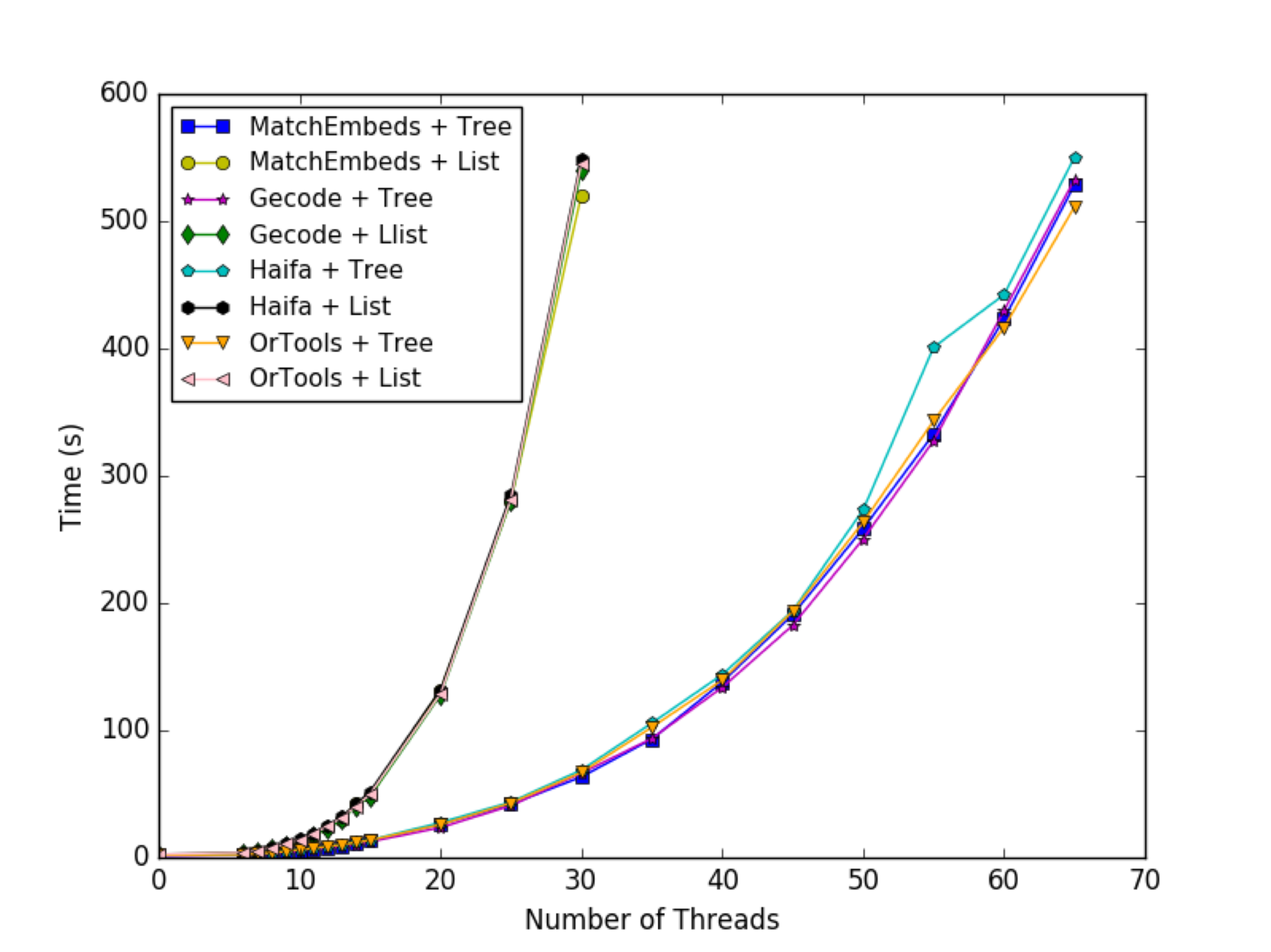
Experiment Count Threads

```
main():
  count = 0
  for i = 1 to N:
    fork thread
  assert(count ≤ N)

thread():
  count = count+1
```



Experiment Secret Sharing

```
main():
 from = 0
 while (*)
  local secret = *
  assume(secret > 0)
  for i = 1 to N:
   to = secret
   fork thread
   while (to > 0): skip
  if (from > 0):
   assert(from == secret)

thread():
  local m = to
  to = 0
  from = m
```



Experiments

- Is Match embeds Practical?
  - Does it improve performance of Proof Spaces?
  - Does the k-d structure improve Proof Spaces?

Experiments

- Is Match embeds Practical?
  - Does it improve performance of Proof Spaces?
  - Does the k-d structure improve Proof Spaces?
- Can MatchEmbeds solve difficult problem instances?

Experiments

- Is Match embeds Practical?
  - Does it improve performance of Proof Spaces?
  - Does the k-d structure improve Proof Spaces?
- Can MatchEmbeds solve difficult problem instances?

- Compared to Constraint Satisfaction Problem Solvers:
  - Gecode - a top competitor in MiniZinc (CSP Competition)
  - HaifaCSP - 1st prize in 2017 MiniZinc competition
  - OrTool's - Google's Optimization/CSP solver

Experiment Random Graphs

- PA emptiness checks lead to “easy” embedding instances

Experiment Random Graphs

- PA emptiness checks lead to “easy” embedding instances
- Generate random “difficult” instances

Experiment Random Graphs

- PA emptiness checks lead to “easy” embedding instances
- Generate random “difficult” instances
  - Generate vocabulary with 2-10 monadic predicates and 1 edge predicate

Experiment Random Graphs

- PA emptiness checks lead to "easy" embedding instances
- Generate random "difficult" instances
  - Generate vocabulary with 2-10 monadic predicates and 1 edge predicate
  - Generate source $\mathfrak{A}$

Experiment Random Graphs

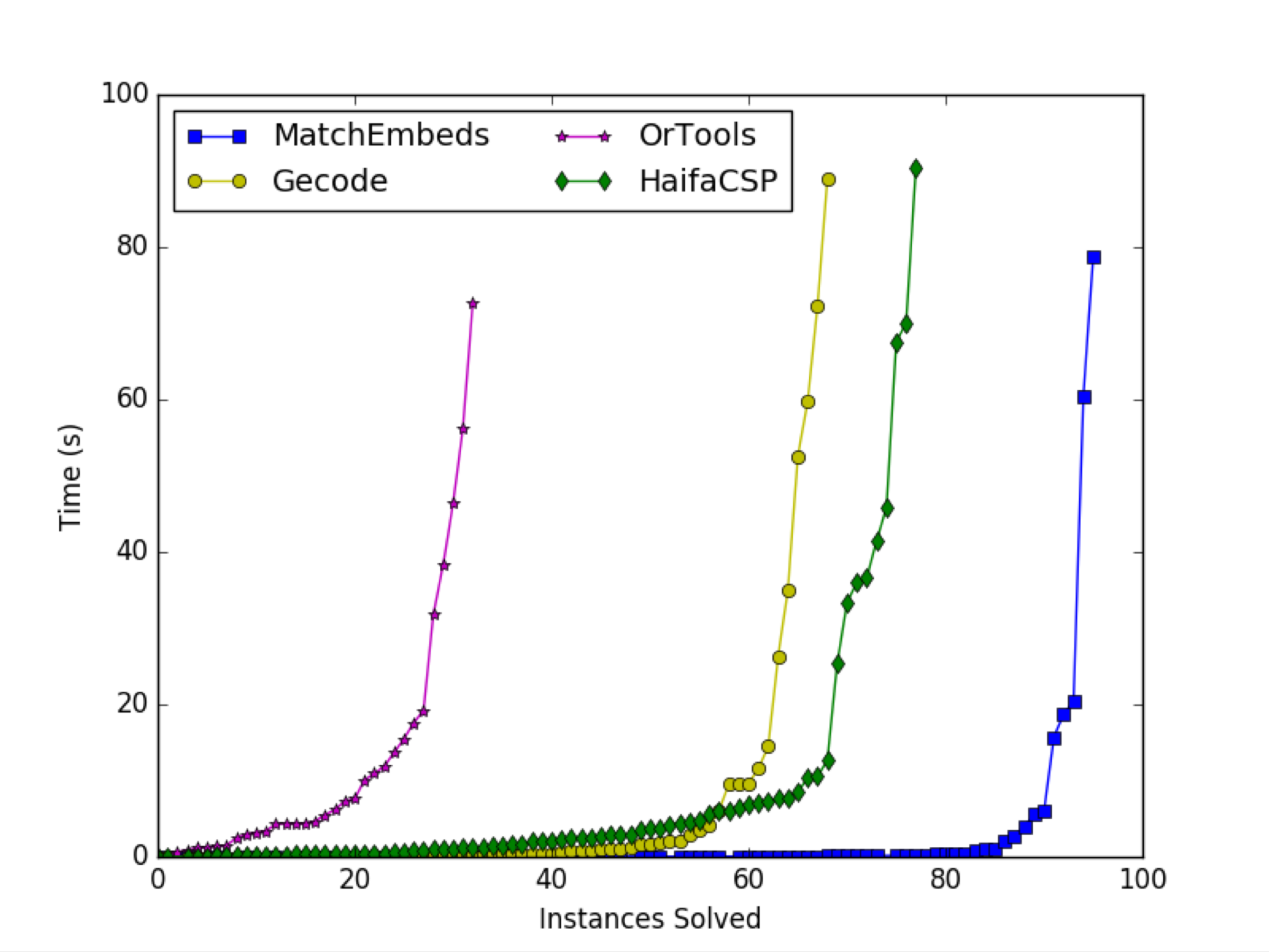
- PA emptiness checks lead to “easy” embedding instances
- Generate random “difficult” instances
  - Generate vocabulary with 2-10 monadic predicates and 1 edge predicate
  - Generate source $\mathfrak{A}$
    - $|A| \in [10,50]$ universe size
    - $p \in [0.1,0.25]$ probability of universe element to appear in monadic predicate
    - $e \in (0,0.1]$ probability of edge between elements

Experiment Random Graphs

- PA emptiness checks lead to “easy” embedding instances
- Generate random “difficult” instances
  - Generate vocabulary with 2-10 monadic predicates and 1 edge predicate
  - Generate source $\mathfrak{A}$
    - $|A| \in [10,50]$ universe size
    - $p \in [0.1,0.25]$ probability of universe element to appear in monadic predicate
    - $e \in (0,0.1]$ probability of edge between elements
  - Generate target $\mathfrak{B}$
    - $|B| \in [|A|, 2|A|]$
    - $p' \in [p, 2p]$
    - $e' \in [e, 4e]$

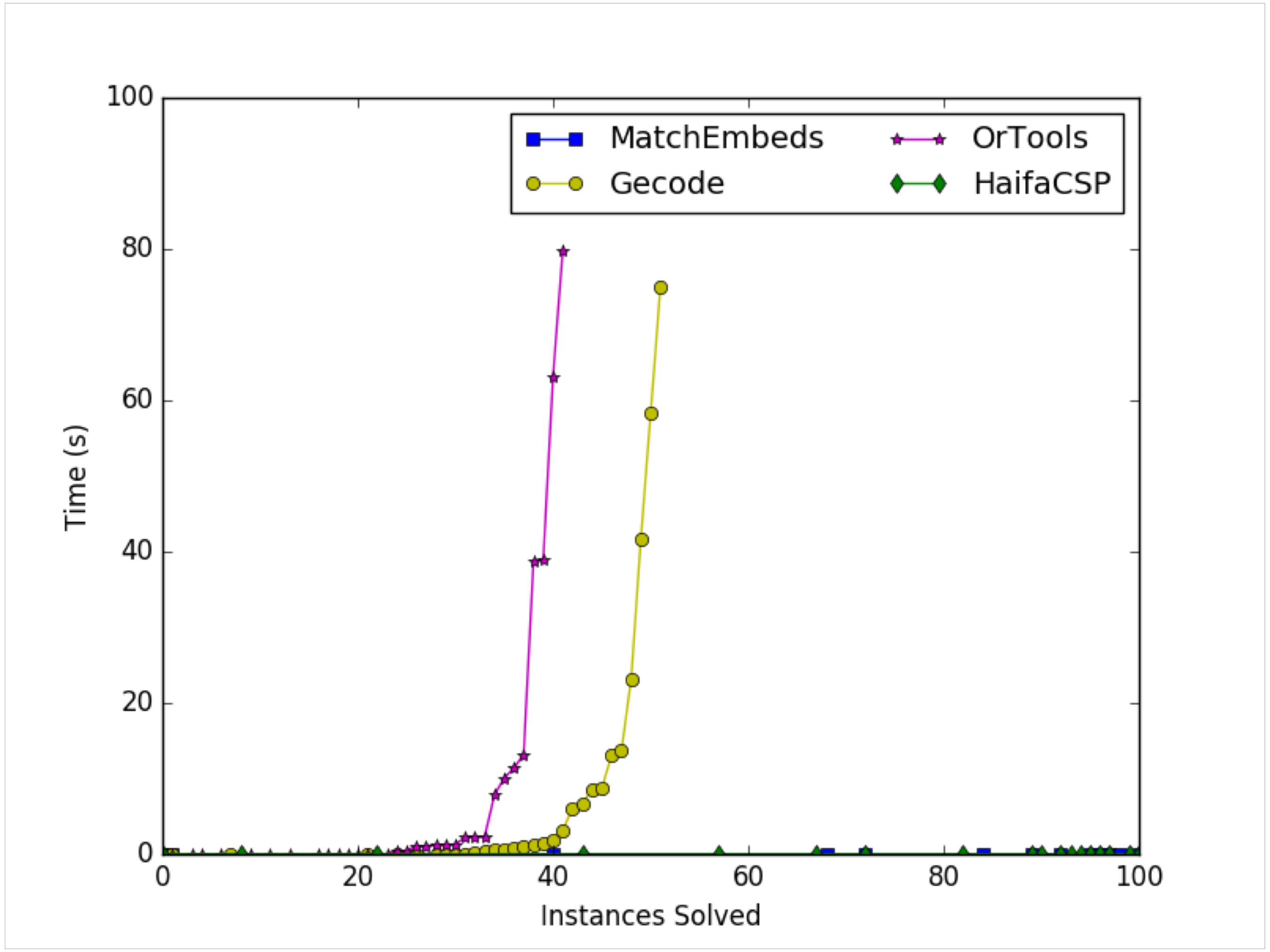
Experiment Random Graphs

- Generate 100 instances
  - 48 positive embeddings
  - 47 negative embeddings
  - 5 unsolved embeddings



Experiment Random Monadic Structures

- Generate 100 instances
  - 56 positive embeddings
  - 44 negative embeddings

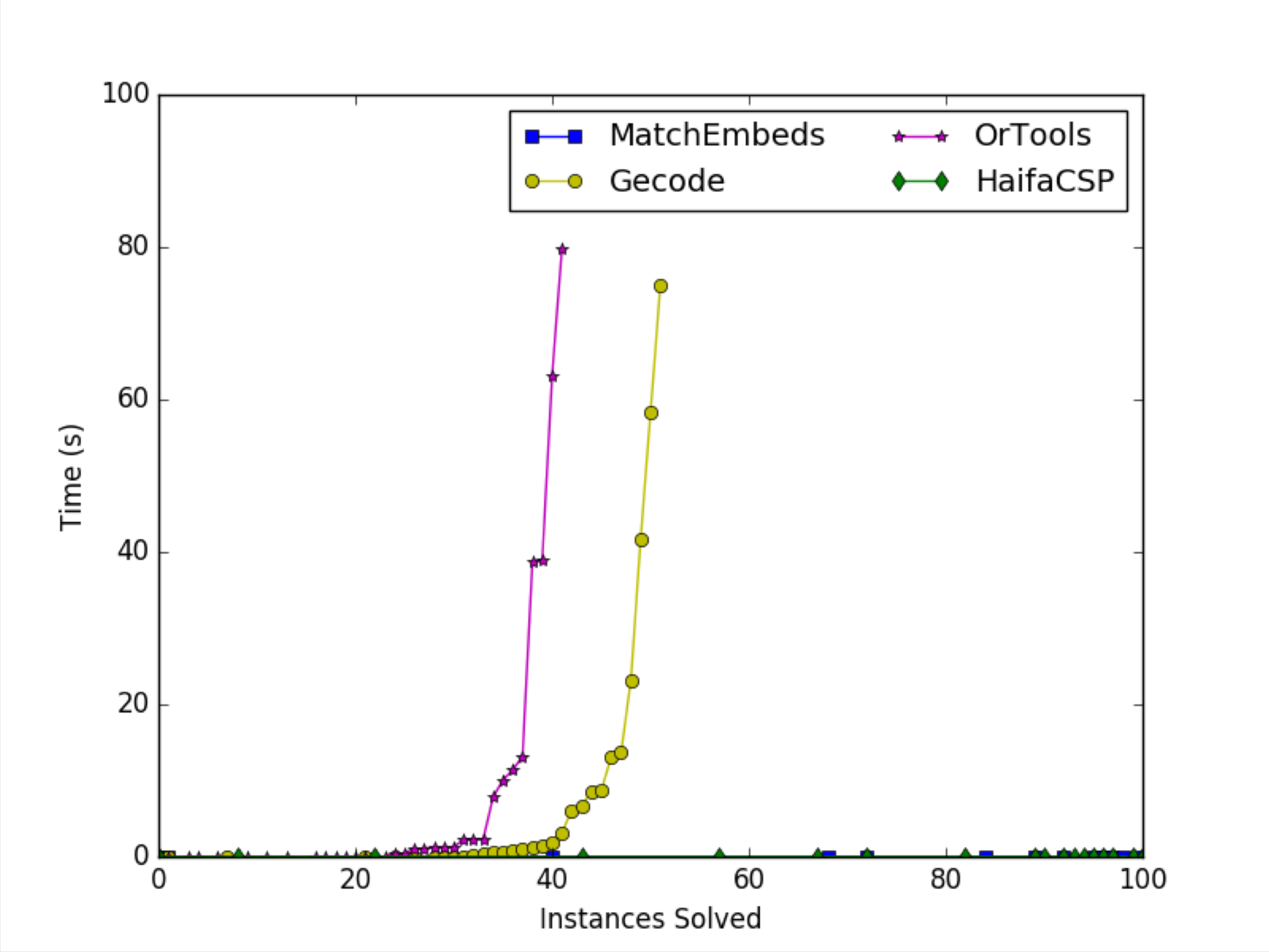


Experiment Random Monadic Structures

- Generate 100 instances
  - 56 positive embeddings
  - 44 negative embeddings

- Match Embeds & HaifaCSP[1]
  - Polytime monadic instances

[Régin, 1994][1]



Related Works

- Régin's Algorithm:

Related Works

- Régin's Algorithm:
  - Constraint of difference (filtering algorithm):

Related Works

- Régin’s Algorithm:
  - Constraint of difference (filtering algorithm):
    1. Remove filtered edges
    2. Compute Maximum Matching
    3. Remove any edges not belonging to maximum matching

Related Works

- Régin's Algorithm:
  - Constraint of difference (filtering algorithm):
    1. Remove filtered edges
    2. Compute Maximum Matching
    3. Remove any edges not belonging to maximum matching
- Sub-graph Isomorphism:

Related Works

- Régin's Algorithm:
  - Constraint of difference (filtering algorithm):
    1. Remove filtered edges
    2. Compute Maximum Matching
    3. Remove any edges not belonging to maximum matching
- Sub-graph Isomorphism:
  - Specialization of structure embedding

Related Works

- Régin’s Algorithm:
  - Constraint of difference (filtering algorithm):
    1. Remove filtered edges
    2. Compute Maximum Matching
    3. Remove any edges not belonging to maximum matching
- Sub-graph Isomorphism:
  - Specialization of structure embedding
  - Focus: find all such isomorphisms

Related Works

- Régin's Algorithm:
  - Constraint of difference (filtering algorithm):
    1. Remove filtered edges
    2. Compute Maximum Matching
    3. Remove any edges not belonging to maximum matching
- Sub-graph Isomorphism:
  - Specialization of structure embedding
  - Focus: find all such isomorphisms
  - Exploit local structure rather than global structure
    - None known to take advantage of all difference constraint

Summary

- MatchEmbeds:
  - Structure Embedding Problem
    - Practical (1-2 orders of magnitude faster than existing solutions)
    - Polytime for monadic instances

Summary

- MatchEmbeds:
  - Structure Embedding Problem
    - Practical (1-2 orders of magnitude faster than existing solutions)
    - Polytime for monadic instances
  - Improves Proof Spaces
    - Verify programs with 70 threads vs 20-30 threads

Summary

- MatchEmbeds:
  - Structure Embedding Problem
    - Practical (1-2 orders of magnitude faster than existing solutions)
    - Polytime for monadic instances
  - Improves Proof Spaces
    - Verify programs with 70 threads vs 20-30 threads
- k-d structure (multi-source embeddings)
  - Avoids unnecessary embeddings
  - Further Improves Proof Spaces
    - Verify programs with 20+ more threads.

References

[1] Kincaid, Z. Podelski, A., Farzan, A. *Proof Spaces for Unbounded Parallelism*. POPL, pgs. 407-420 (2015).

[2] Finkel, A. Schnoebelen, Ph. *Well Structured Transition Systems Everywhere.* Theoretical Computer Science Vol 256:1, pgs. 63-92 (2001).

[3] Hopcroft, J., Karp, R. *An* $n^{5/2}$ *Algorithm for Maximum Matchings in Bipartite Graphs*. SIAM Journal of Computing, Vol. 2, No. 5 : pgs. 225-231 (1973).

[4] Régin, J.C.: *A filtering Algorithm for Constraints of Difference in CSPs*. In: AAAI. pgs. 362-367 (1994)

[5] Russell, S.J., Norvig, P. *Artificial Intelligence - a Modern Approach*, 3rd Edition. Prentice Hall series in Artificial Intelligence, Prentice Hall (2009)

Extra Slides

Proof Spaces

[Kincaid et. al. 2015]

Multi-Threaded Program Verification

- Unbounded number of threads

Multi-Threaded Program Verification

- Unbounded number of threads
  - Webservers, databases, computations over *N* threads

Multi-Threaded Program Verification

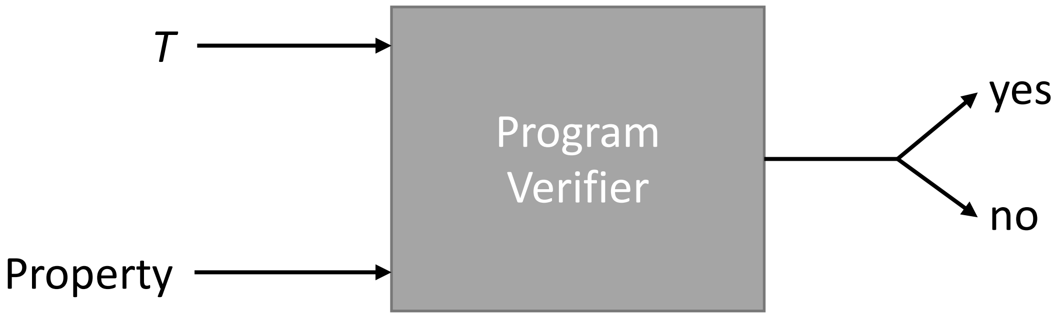
- Unbounded number of threads
  - Webservers, databases, computations over *N* threads
  - Uses single template *T* executed by each thread

$$T^N = \underbrace{T \parallel T \parallel \cdots \parallel T}_{N \text{ times}}$$

Multi-Threaded Program Verification

- Unbounded number of threads
  - Webservers, databases, computations over *N* threads
  - Uses single template *T* executed by each thread

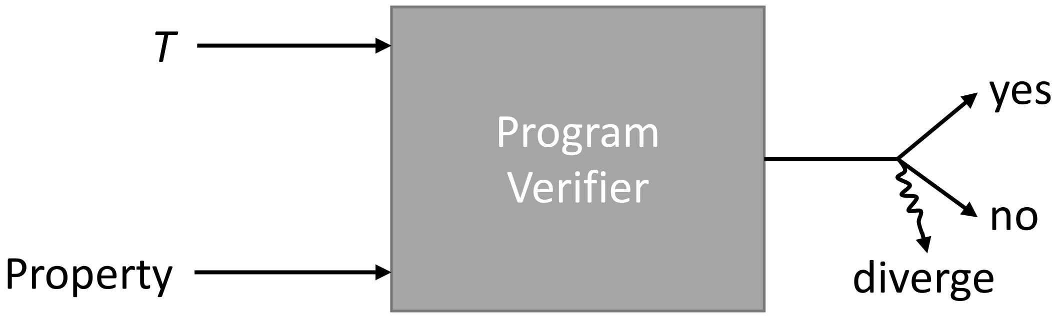
$$T^N = \underbrace{T \parallel T \parallel \cdots \parallel T}_{N \text{ times}}$$



Multi-Threaded Program Verification

- Unbounded number of threads
  - Webservers, databases, computations over *N* threads
  - Uses single template *T* executed by each thread

$$T^N = \underbrace{T \parallel T \parallel \cdots \parallel T}_{N \text{ times}}$$



Multi-threaded Program Verification

- Key Ideas:

Multi-threaded Program Verification

- Key Ideas:
  - Multi-threaded verification is hard

Multi-threaded Program Verification

- Key Ideas:
  - Multi-threaded verification is hard
  - Verify individual traces
    - Reuse sequential verification

Multi-threaded Program Verification

- Key Ideas:
  - Multi-threaded verification is hard
  - Verify individual traces
    - Reuse sequential verification

Program P is correct $\Leftrightarrow$ all traces of P are correct

Multi-threaded Program Verification

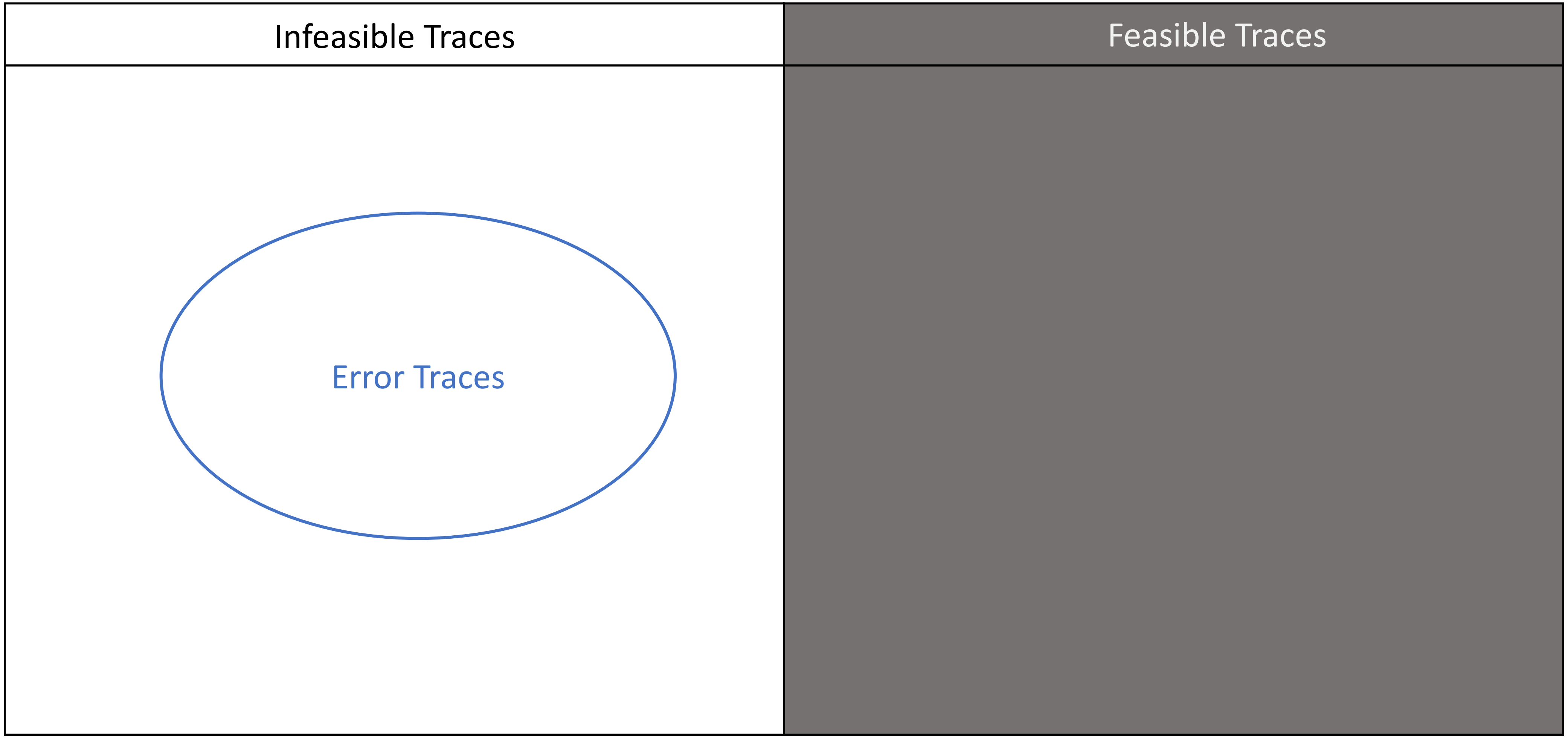
- Key Ideas:
  - Multi-threaded verification is hard
  - Verify individual traces
    - Reuse sequential verification

Program P is correct $\Leftrightarrow$ all traces of P are correct

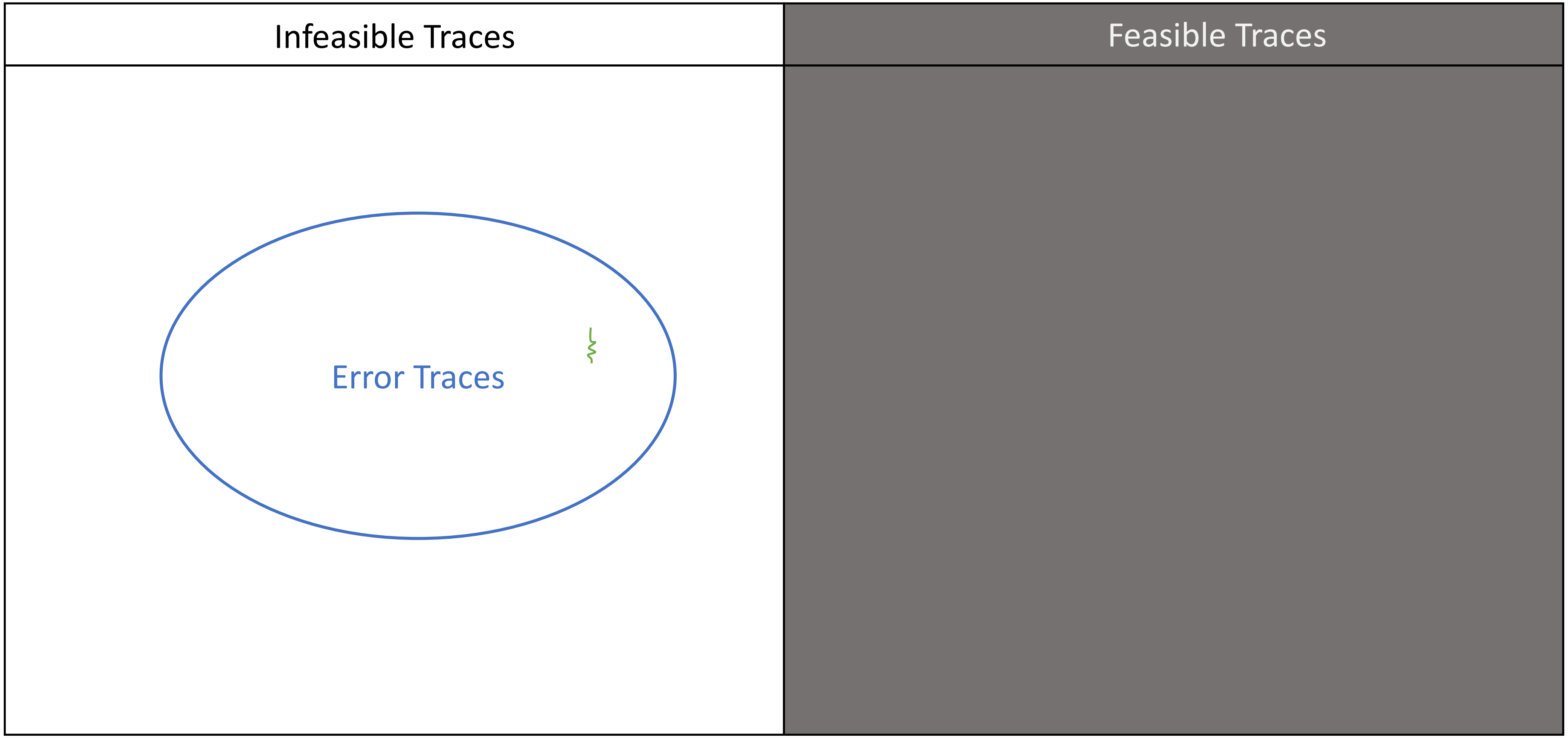
- Focus:

$$P = T^N = \underbrace{T \parallel T \parallel \cdots \parallel T}_{N \text{ times}}$$

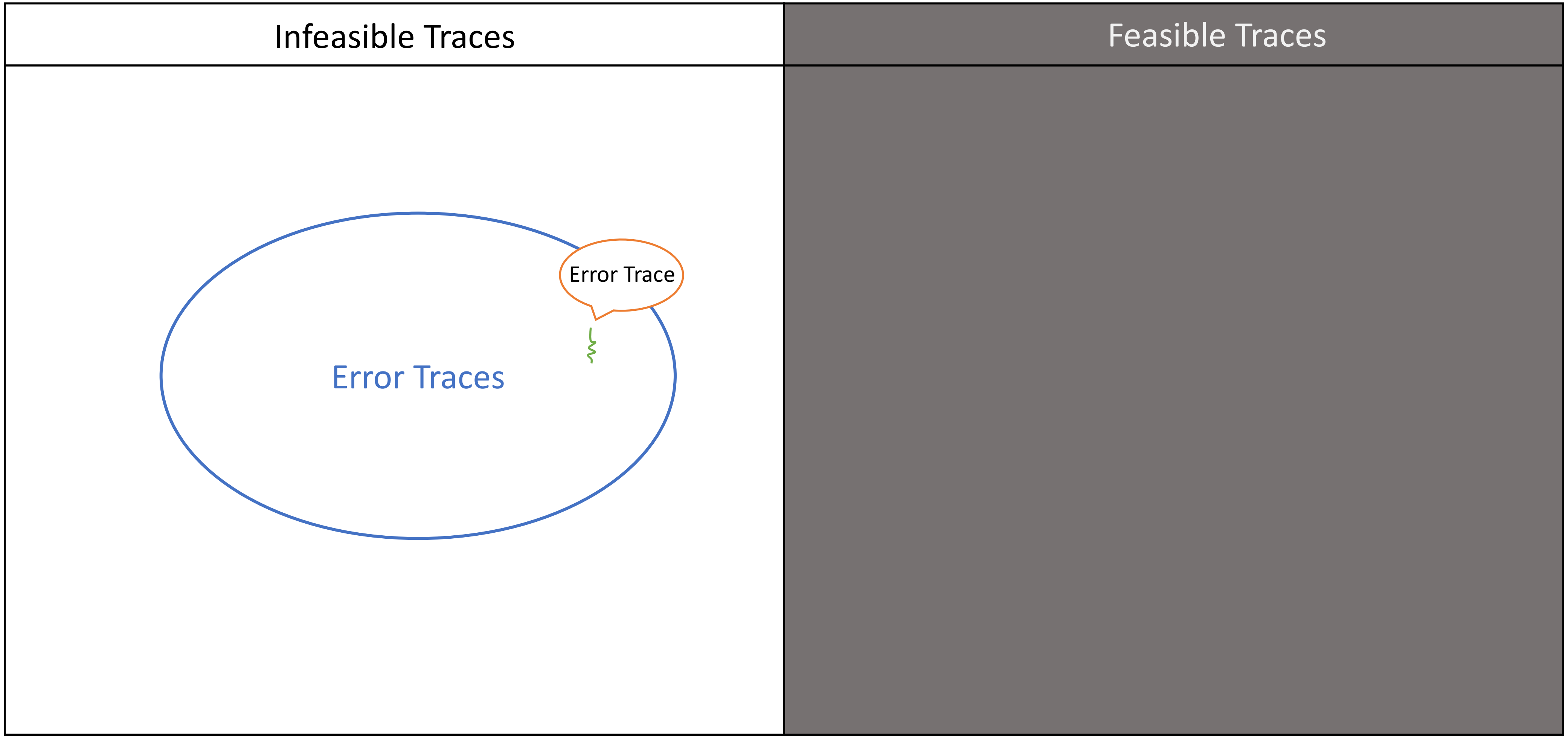
Multi-threaded Program Verification



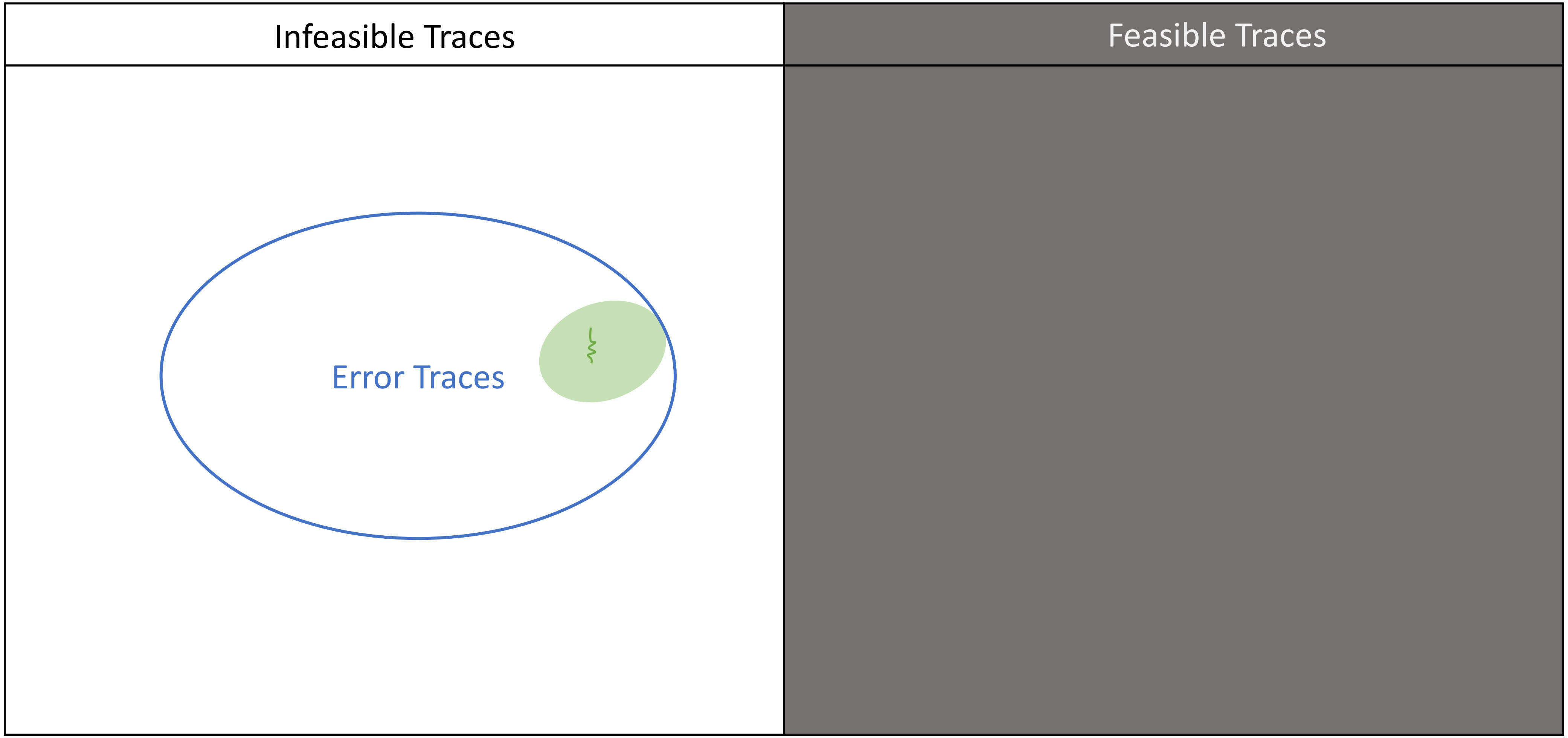
Multi-threaded Program Verification



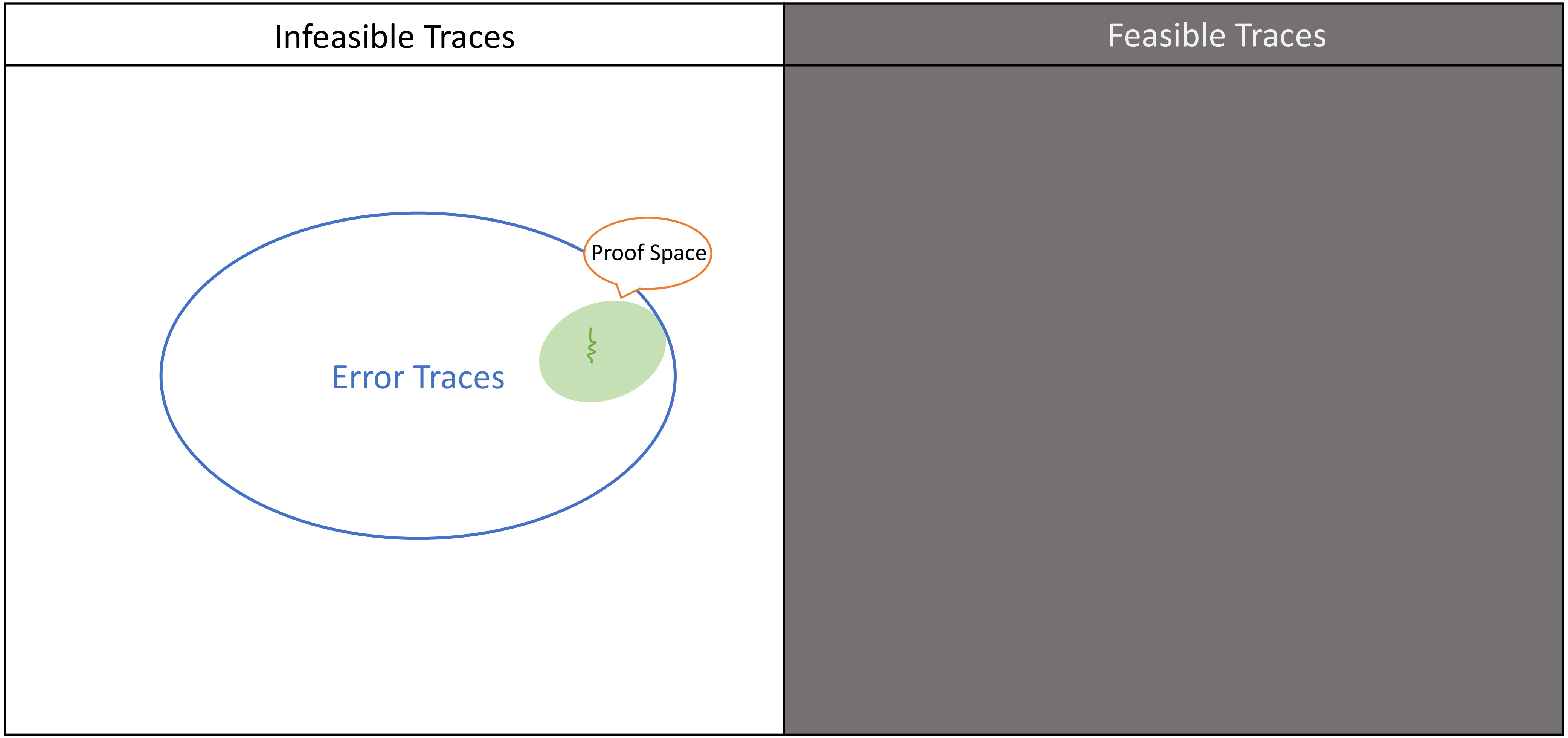
Multi-threaded Program Verification



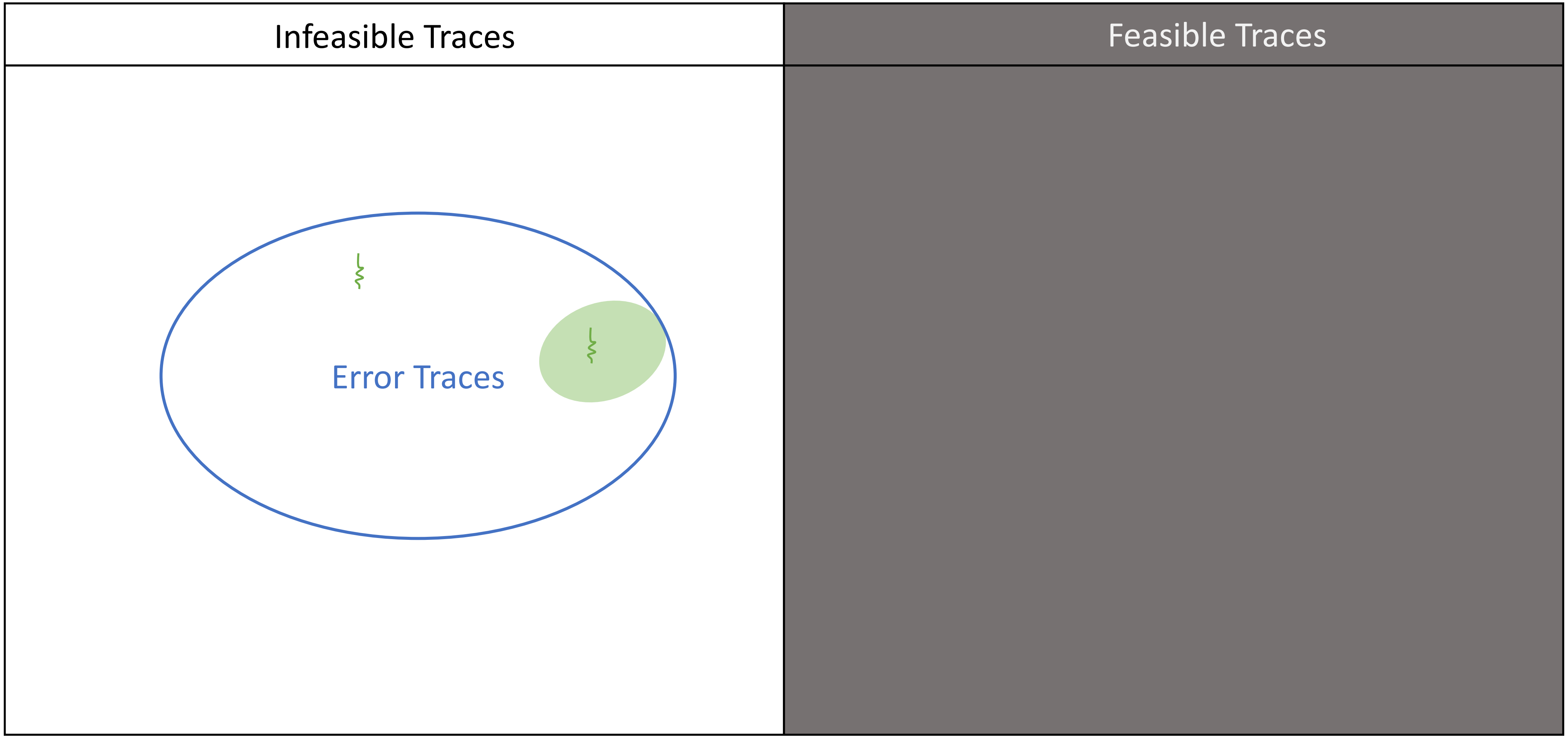
Multi-threaded Program Verification



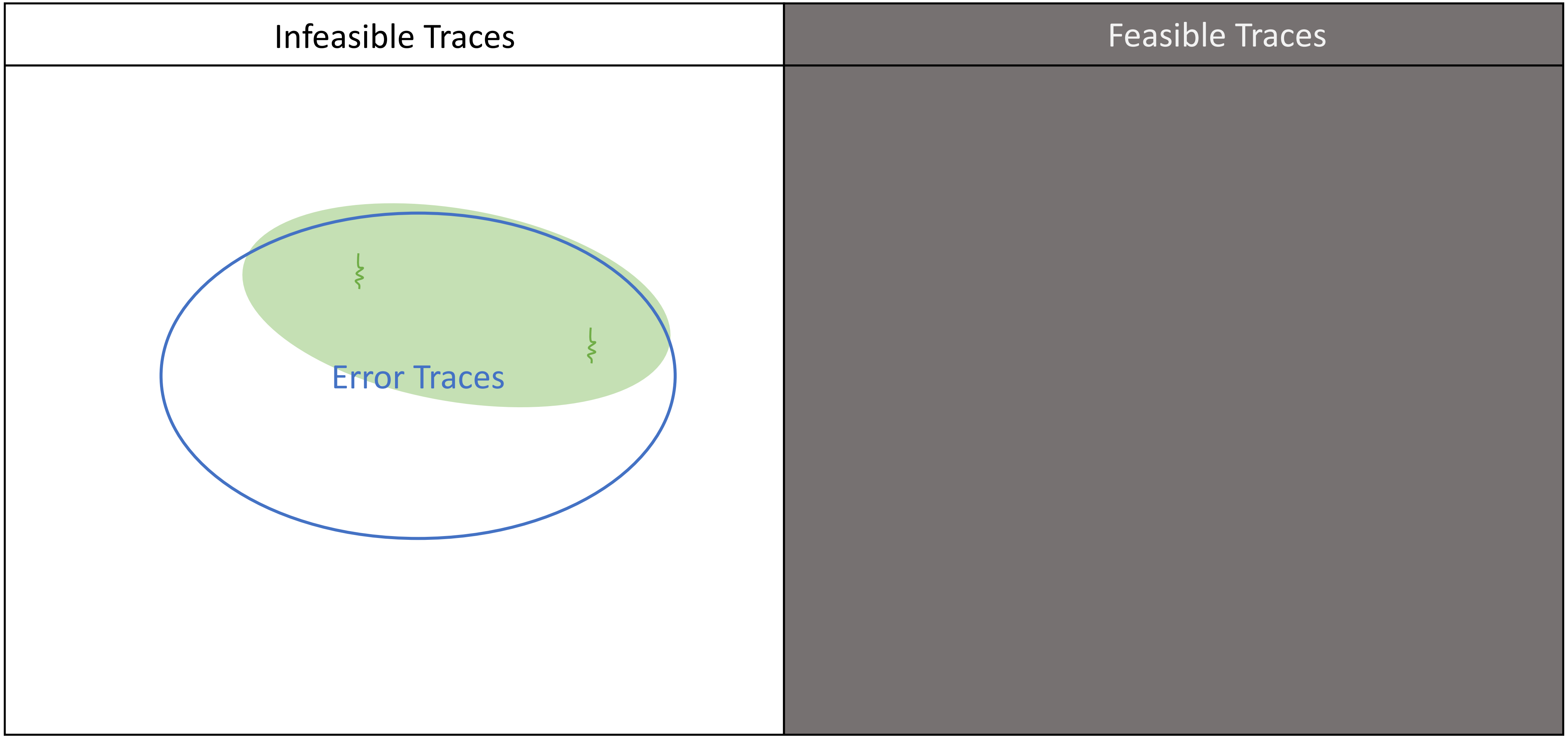
Multi-threaded Program Verification



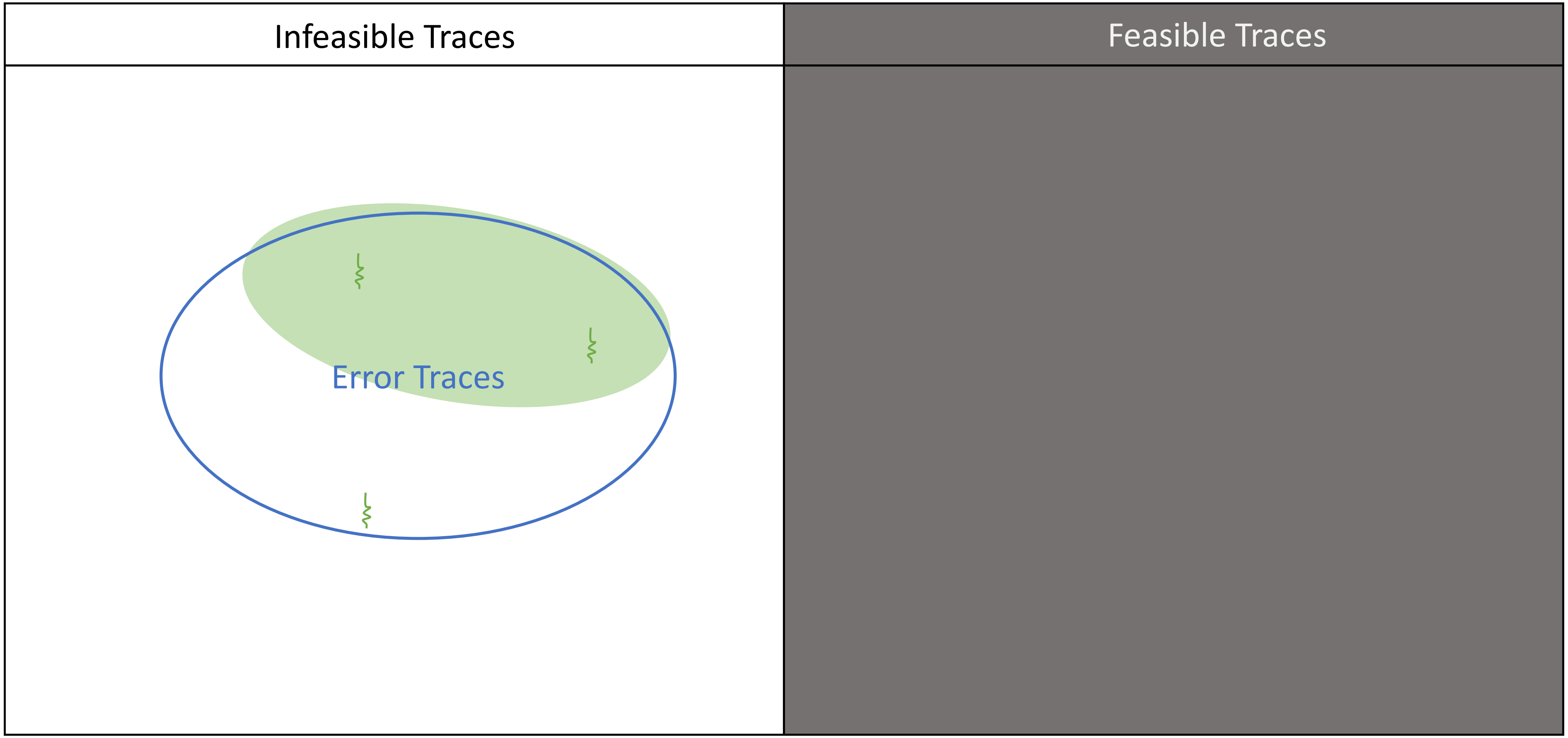
Multi-threaded Program Verification



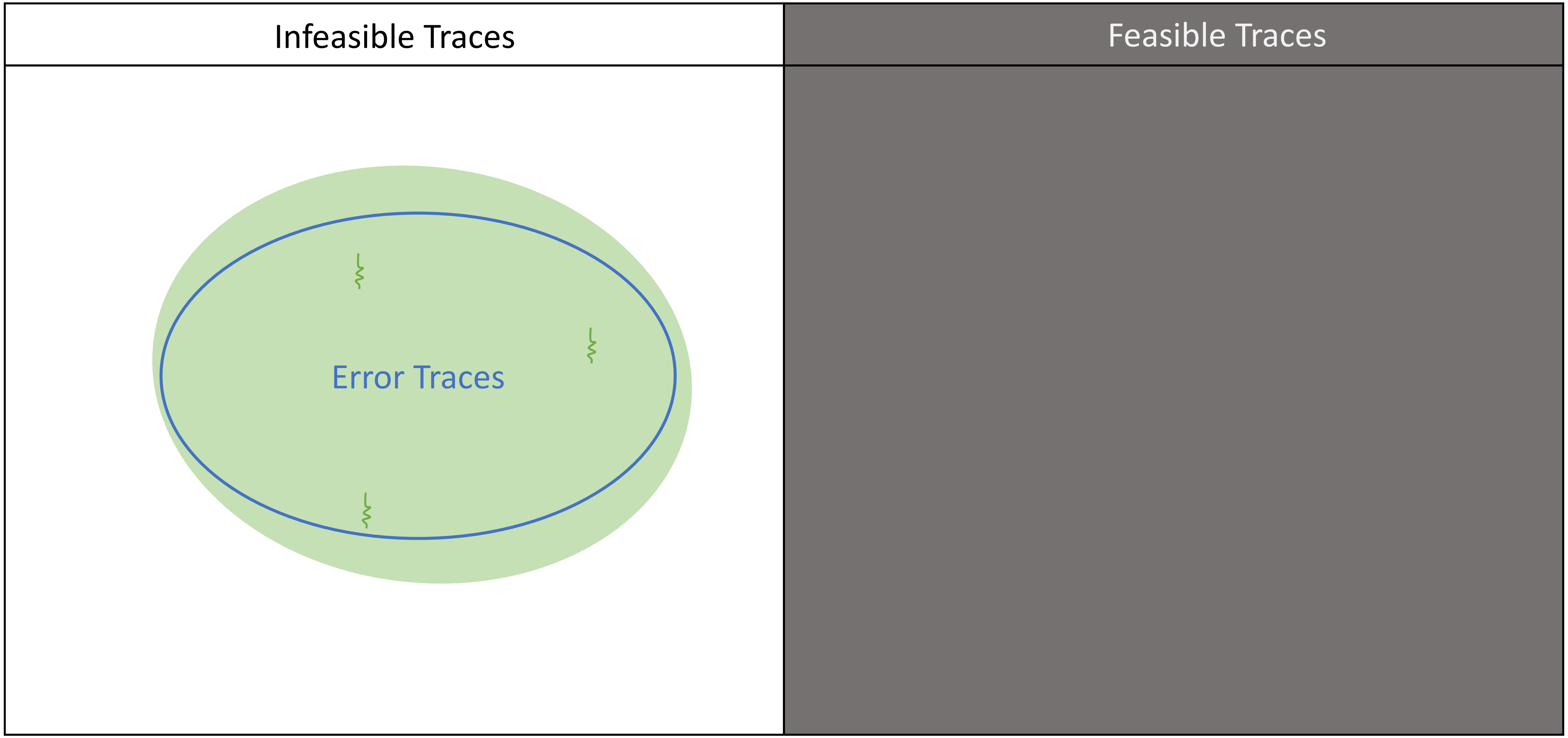
Multi-threaded Program Verification



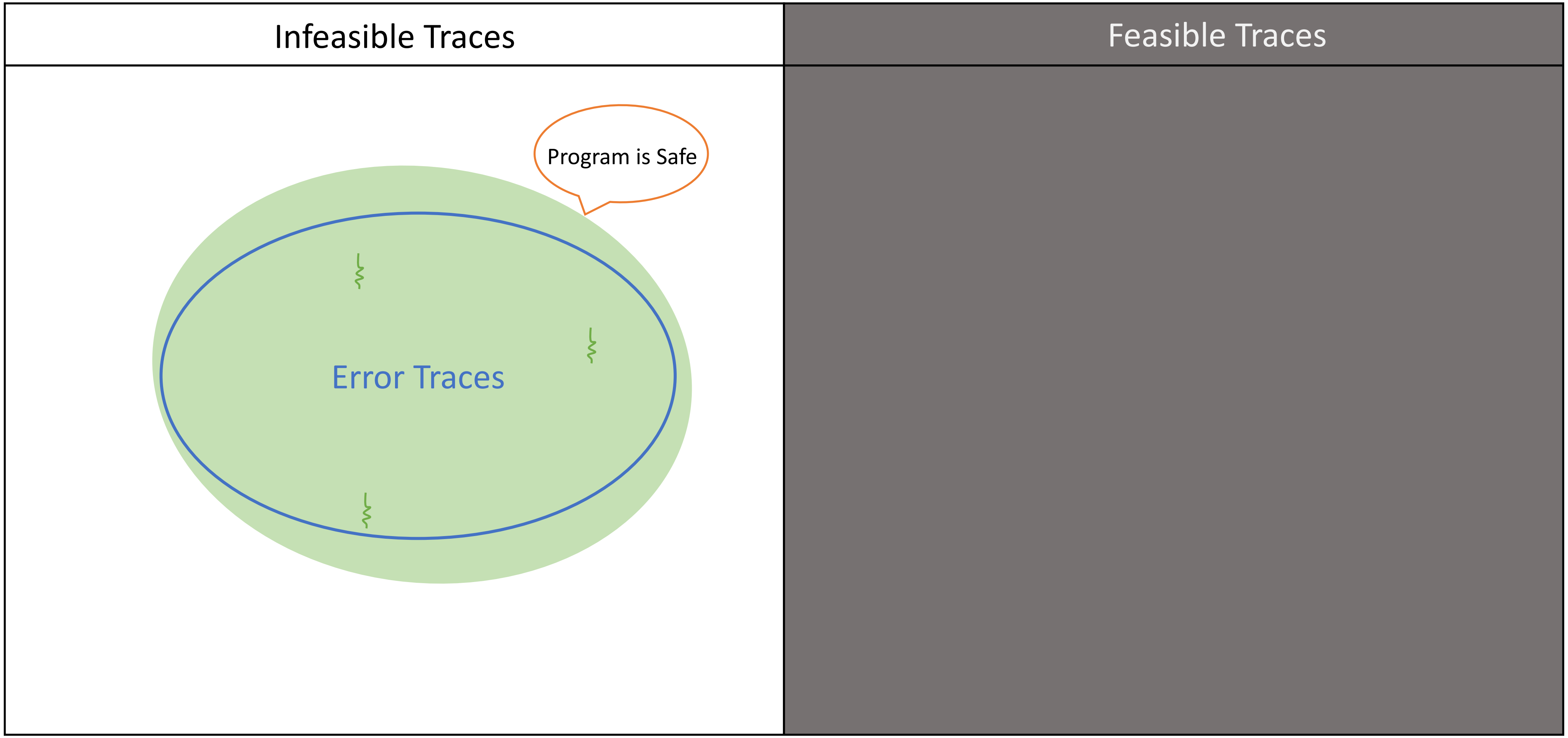
Multi-threaded Program Verification



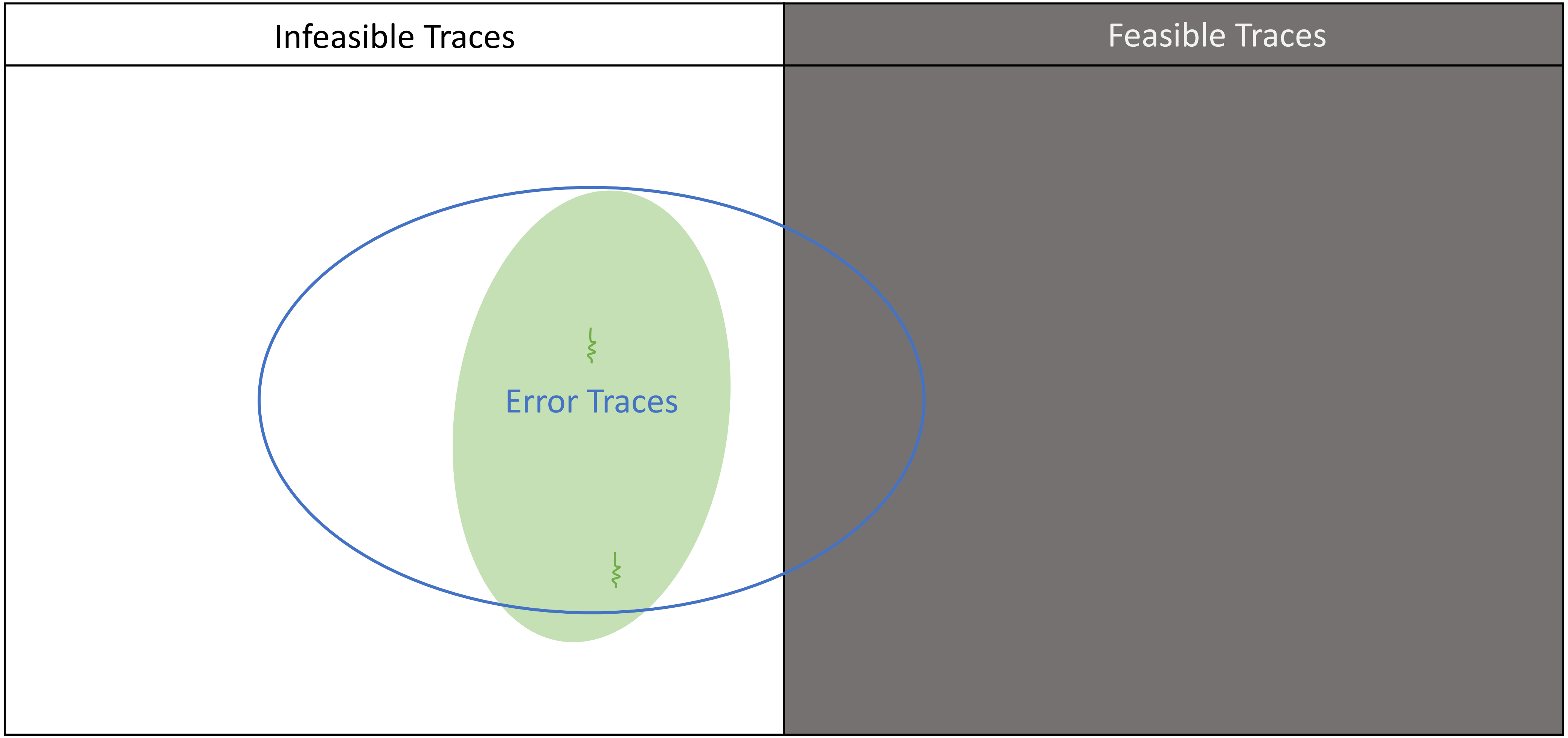
Multi-threaded Program Verification



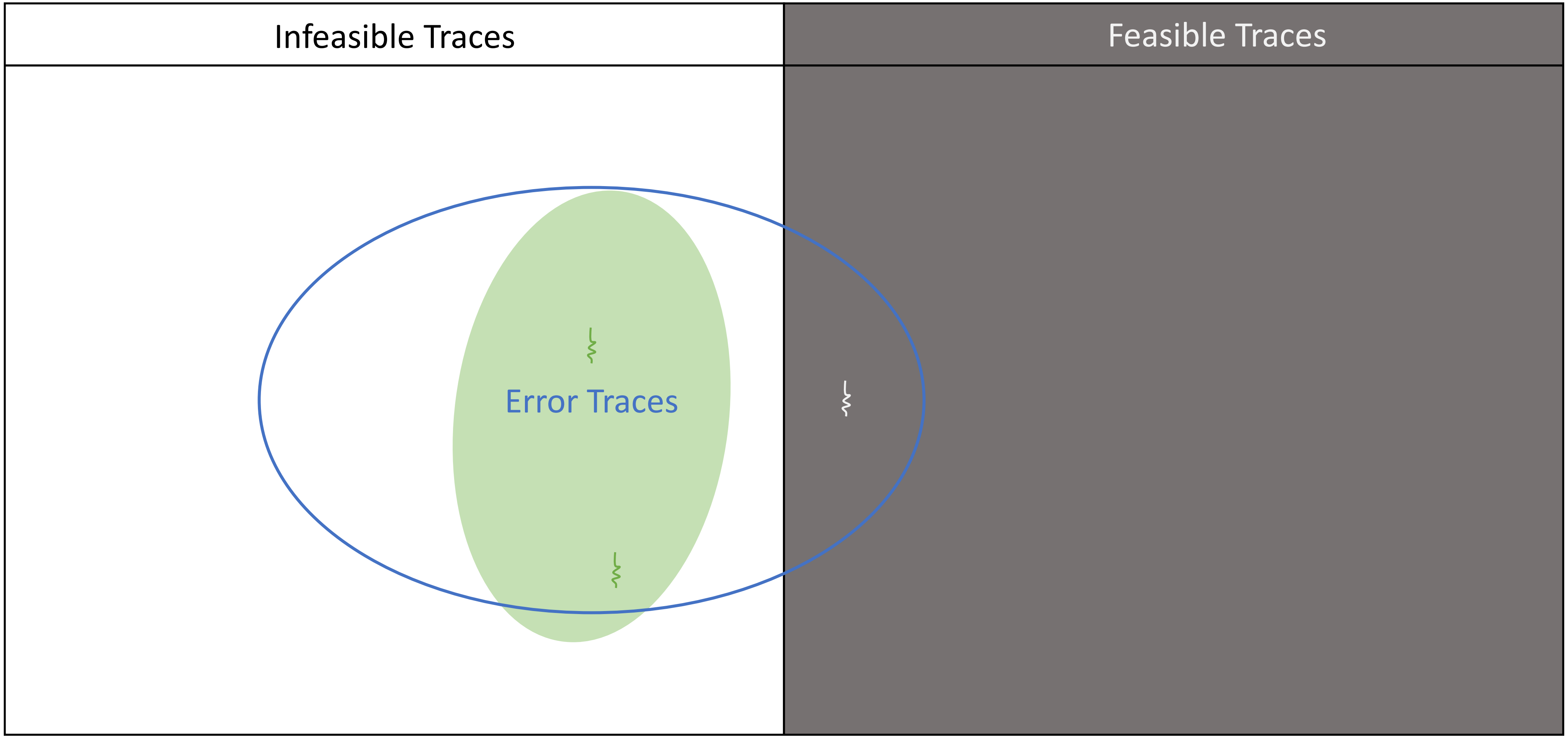
Multi-threaded Program Verification



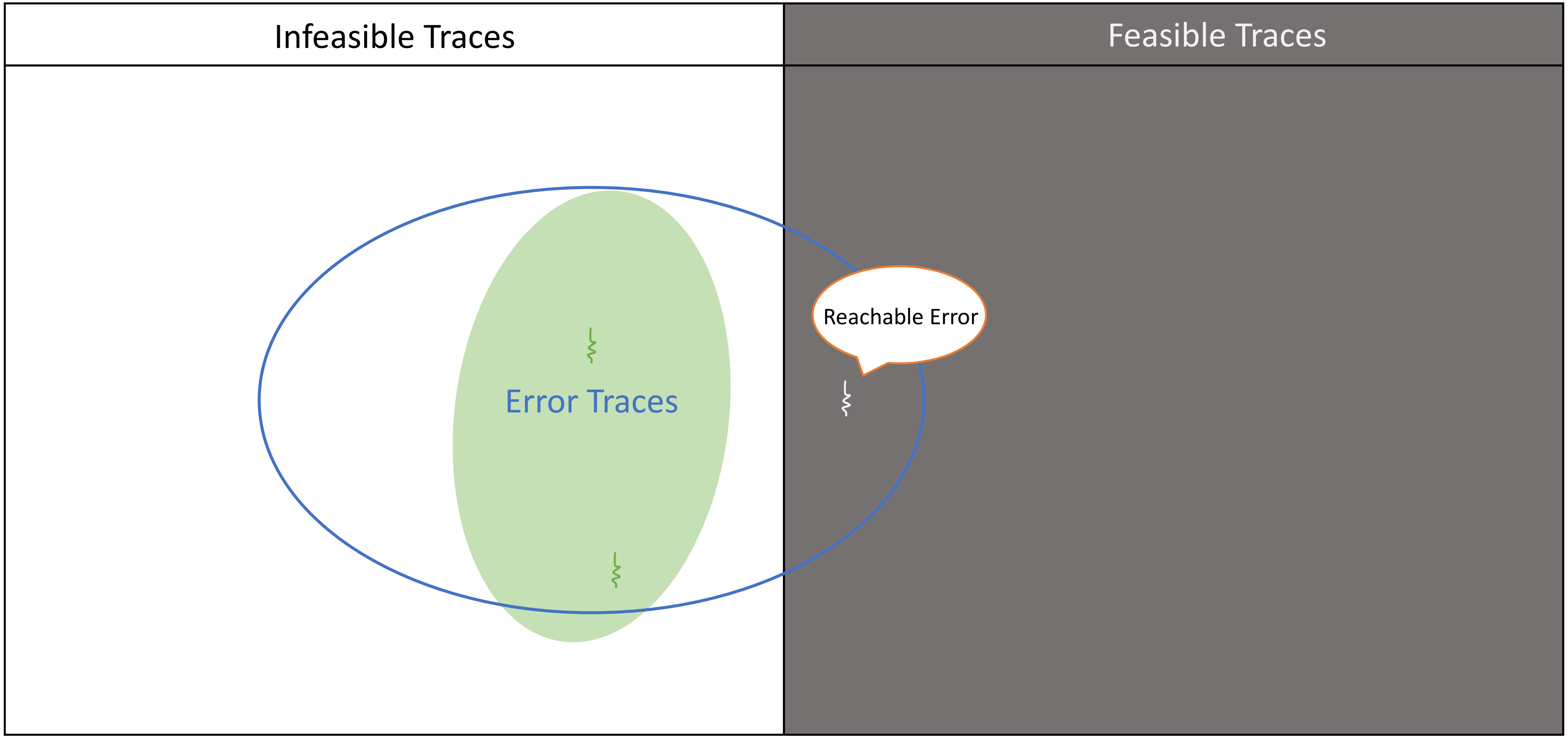
Multi-threaded Program Verification



Multi-threaded Program Verification



Multi-threaded Program Verification



Proof Spaces

- A **proof space** is a **valid** set of Hoare triples

Proof Spaces

- A **proof space** is a **valid** set of Hoare triples
  - Closed under sequencing

Proof Spaces

- A **proof space** is a **valid** set of Hoare triples
  - Closed under sequencing

$$\frac{\{P(a_1, \dots, a_{ar(P)})\}\ C\!: t\ \{Q(b_1, \dots, b_{ar(Q)})\} \qquad \{Q(b_1, \dots, b_{ar(Q)})\}\ C'\!: s\ \{R(c_1, \dots, c_{ar(R)})\}}{\{P(a_1, \dots, a_{ar(P)})\}\ C\!: t; C'\!: s\ \{R(c_1, \dots, c_{ar(R)})\}}\ (seq)$$

Proof Spaces

- A **proof space** is a **valid** set of Hoare triples
  - Closed under sequencing, symmetry

Proof Spaces

- A **proof space** is a **valid** set of Hoare triples
  - Closed under sequencing, symmetry

$$\frac{\pi\colon\ \mathbb{N} \to \mathbb{N} \text{ is a permutation} \qquad \left\{P(a_1, \dots, a_{ar(P)})\right\}\, C{:}\, t\, \left\{Q(b_1, \dots, b_{ar(R)})\right\}}{\left\{P(\pi(a_1), \dots, \pi(a_{ar(P)}))\right\}\, C{:}\, \pi(t)\, \left\{Q(\pi(b_1), \dots, \pi(b_{ar(Q)}))\right\}}\ (symm)$$

Proof Spaces

- A **proof space** is a **valid** set of Hoare triples
  - Closed under sequencing, symmetry, conjunction

Proof Spaces

- A **proof space** is a **valid** set of Hoare triples
  - Closed under sequencing, symmetry, conjunction

$$\frac{\{P(a_1, \dots, a_{ar(P)})\}\ C{:}\,t\ \{Q(b_1, \dots, b_{ar(Q)})\} \qquad \{R(c_1, \dots, c_{ar(R)})\}\ C{:}\,t\ \{S(d_1, \dots, d_{ar(S)})\}}{\{P(a_1, \dots, a_{ar(P)}) \wedge R(c_1, \dots, c_{ar(R)})\}\ C{:}\,t\ \{Q(b_1, \dots, b_{ar(Q)}) \wedge S(d_1, \dots, d_{ar(S)})\}}\ (conj)$$

Proof Spaces

- A **proof space** is a **valid** set of Hoare triples
  - Closed under sequencing, symmetry, conjunction
  - Generated from a finite "basis" of Hoare triples

Proof Spaces

- A **proof space** is a **valid** set of Hoare triples
  - Closed under sequencing, symmetry, conjunction
  - Generated from a finite “basis” of Hoare triples

If a proof space, H, exists such that for every error trace, τ,

$$\{\text{pre}\}\ \tau\ \{\text{false}\} \in H$$

then the program is safe.

Proof Checking

- For any Proof Space, H,
  - $\{\tau : \{pre\}\,\tau\,\{false\} \in H\}$ is recognized by a Predicate Automata, $A(H)$

Proof Checking

- For any Proof Space, H,
  - $\{\tau : \{pre\}\,\tau\,\{false\} \in H\}$ is recognized by a Predicate Automata, $A(H)$
- For any Program, P,
  - The set of error traces of P is recognized by a PA, Err

Proof Checking

- For any Proof Space, H,
  - $\{\tau : \{pre\}\, \tau\, \{false\} \in H\}$ is recognized by a Predicate Automata, $A(H)$
- For any Program, P,
  - The set of error traces of P is recognized by a PA, Err
- PA languages are closed under intersection and complement

Proof Checking

- For any Proof Space, H,
  - $\{\tau : \{pre\}\,\tau\,\{false\} \in H\}$ is recognized by a Predicate Automata, $A(H)$
- For any Program, P,
  - The set of error traces of P is recognized by a PA, Err
- PA languages are closed under intersection and complement

Proof space inclusion then reduces to PA emptiness:

$$\forall \tau \in \text{Error Trace}.\,\{pre\}\,\tau\,\{false\} \in H$$
$$\Leftrightarrow$$
$$Err \cap \overline{A(H)} = \emptyset$$

Predicate Automata

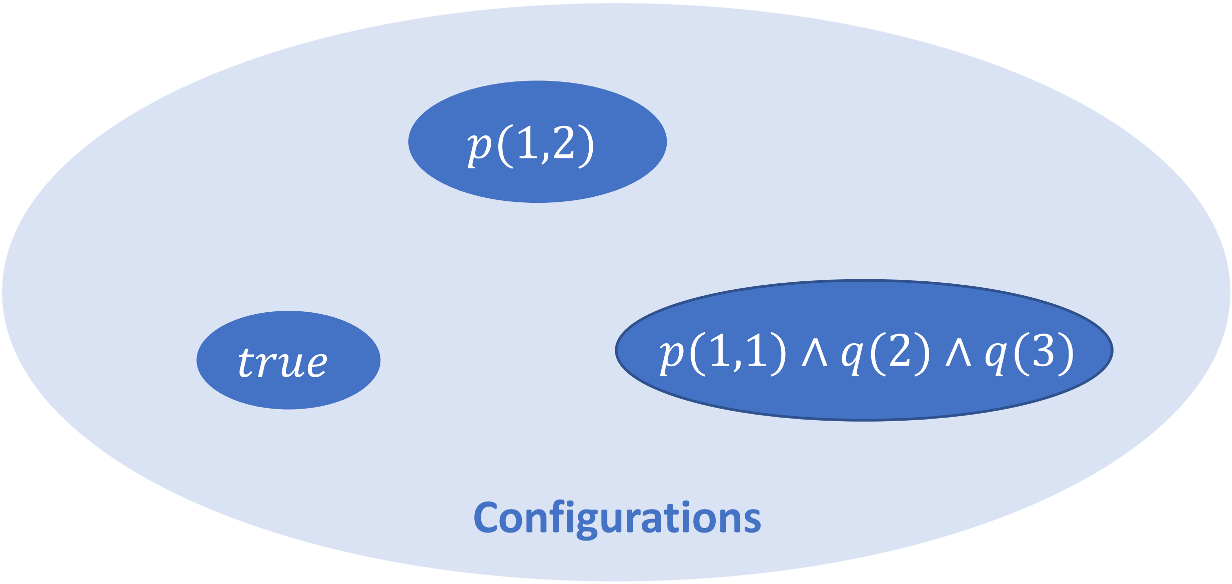
- **Relational vocabulary** $\langle Q, ar \rangle$

$$Q = \{p, q\}, ar(p) = 2, ar(q) = 1$$

Predicate Automata

- **Relational vocabulary** $\langle Q, ar \rangle$

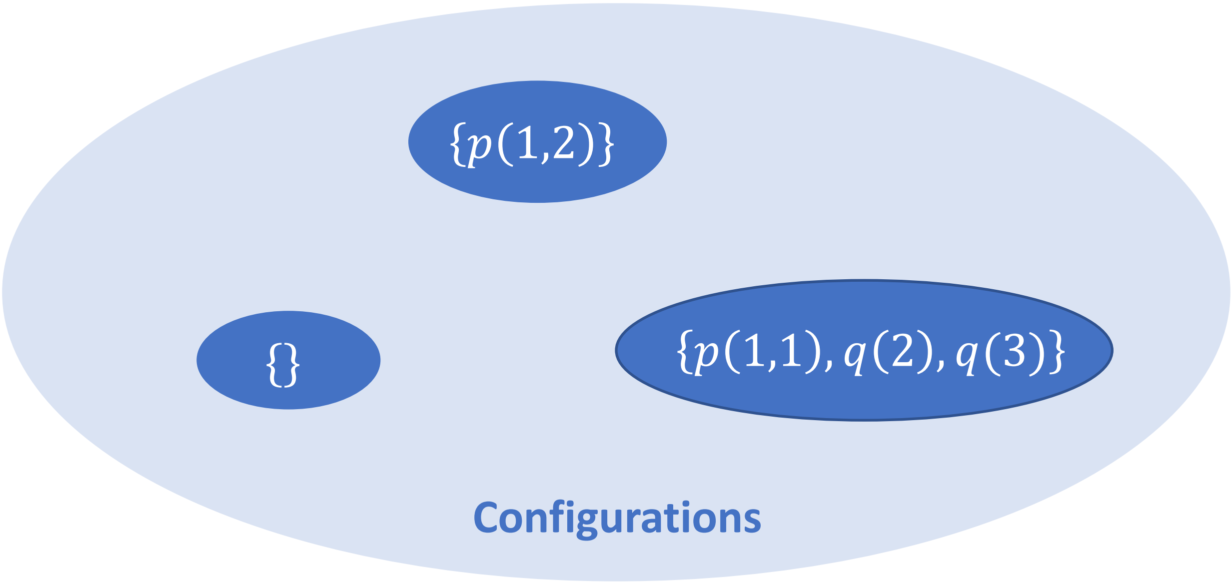
$$Q = \{p, q\}, ar(p) = 2, ar(q) = 1$$



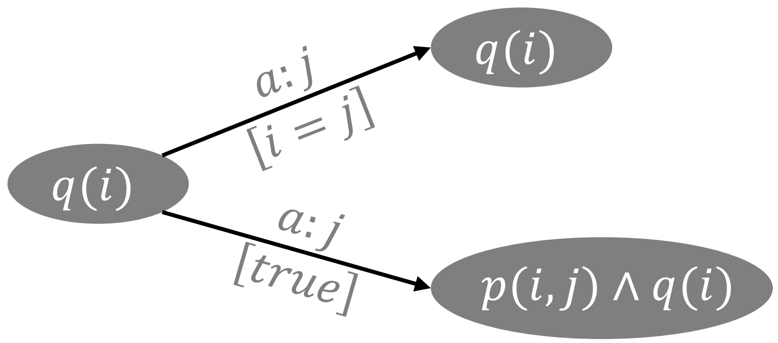
Predicate Automata

- **Relational vocabulary** $\langle Q, ar\rangle$

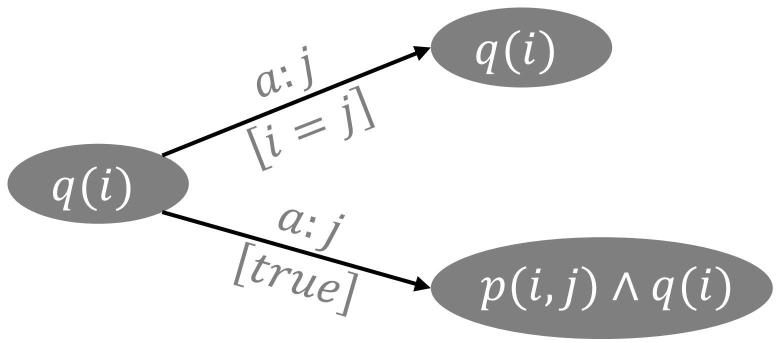
$$Q = \{p, q\}, ar(p) = 2, ar(q) = 1$$

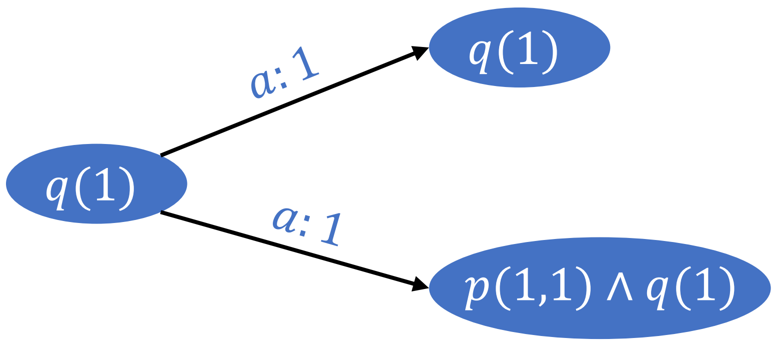


Predicate Automata

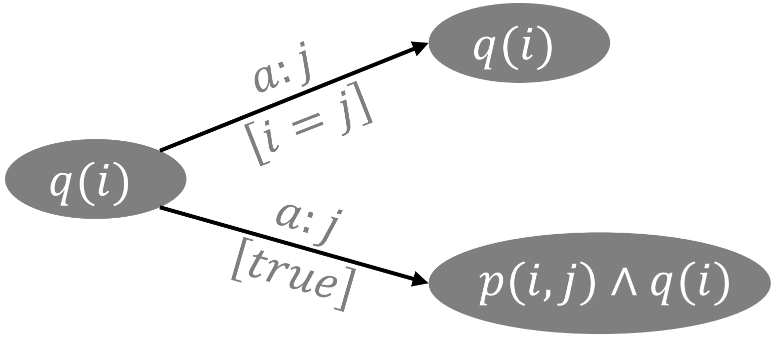


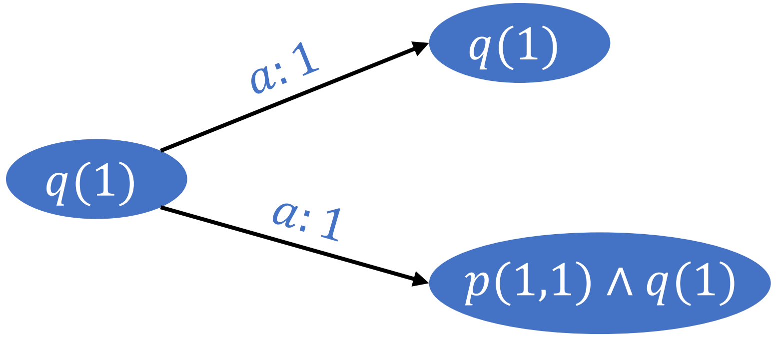
Predicate Automata

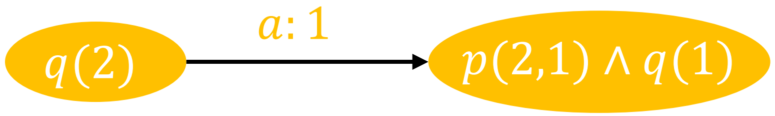




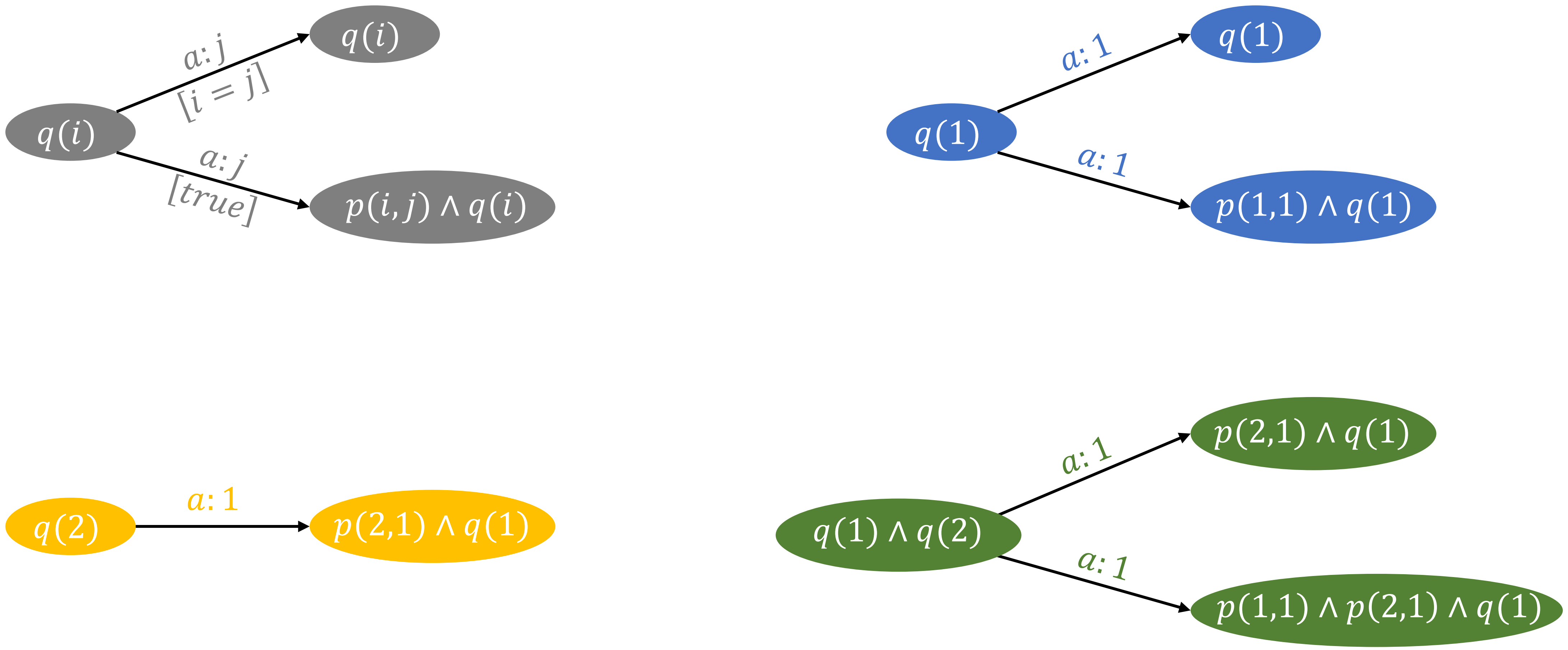
Predicate Automata







Predicate Automata



Predicate Automata

- Infinite State Automata over Infinite Alphabet ($\Sigma \times \mathbb{N}$)
- A = $\langle Q, ar, \Sigma, \delta, \varphi_{start}, F \rangle$

Predicate Automata

- Infinite State Automata over Infinite Alphabet ($\Sigma \times \mathbb{N}$)
- A = $\langle Q, ar, \Sigma, \delta, \varphi_{start}, F \rangle$
  - $\langle Q, ar \rangle$: Relational vocabulary
    - Q : Finite set of predicate symbols
    - $\mathrm{ar} : \mathrm{Q} \rightarrow \mathbb{N}$

Predicate Automata

- Infinite State Automata over Infinite Alphabet ($\Sigma \times \mathbb{N}$)
- A = $\langle Q, ar, \Sigma, \delta, \varphi_{start}, F \rangle$
  - $\langle Q, ar \rangle$: Relational vocabulary
    - Q : Finite set of predicate symbols
    - $\mathrm{ar} : \mathrm{Q} \rightarrow \mathbb{N}$
  - Σ : Finite set of letters

Predicate Automata

- Infinite State Automata over Infinite Alphabet ($\Sigma \times \mathbb{N}$)
- A = $\langle Q, ar, \Sigma, \delta, \varphi_{start}, F \rangle$
  - $\langle Q, ar \rangle$: Relational vocabulary
    - Q : Finite set of predicate symbols
    - $\mathrm{ar} : \mathrm{Q} \rightarrow \mathbb{N}$
  - Σ : Finite set of letters
  - $\varphi_{start} \in \mathcal{F}(Q, ar)$: Initial formula (with no free variables)

Predicate Automata

- Infinite State Automata over Infinite Alphabet ($\Sigma \times \mathbb{N}$)
- A = $\langle Q, ar, \Sigma, \delta, \varphi_{start}, F \rangle$
  - $\langle Q, ar \rangle$: Relational vocabulary
    - Q : Finite set of predicate symbols
    - $\mathrm{ar} : \mathrm{Q} \rightarrow \mathbb{N}$
  - Σ : Finite set of letters
  - $\varphi_{start} \in \mathcal{F}(Q, ar)$: Initial formula (with no free variables)
  - $F \subseteq \mathrm{Q}$: Set of accepting predicate symbols.

Predicate Automata

- Infinite State Automata over Infinite Alphabet ($\Sigma \times \mathbb{N}$)
- A = $\langle Q, ar, \Sigma, \delta, \varphi_{start}, F \rangle$
    - $\langle Q, ar \rangle$: Relational vocabulary
        - Q : Finite set of predicate symbols
        - $\mathrm{ar} : \mathrm{Q} \rightarrow \mathbb{N}$
    - Σ : Finite set of letters
    - $\varphi_{start} \in \mathcal{F}(Q, ar)$: Initial formula (with no free variables)
    - $F \subseteq \mathrm{Q}$: Set of accepting predicate symbols.
    - $\delta : \mathrm{Q} \times \Sigma \rightarrow \mathcal{F}(Q, ar)$ the only free variables of $\delta(q, \sigma)$ are the free variables of q and σ

Emptiness Algorithm

$Closed \leftarrow \emptyset$
$N \leftarrow \emptyset$
$E \leftarrow \emptyset$
$wl \leftarrow dnf(\varphi_{start})$
while $wl \neq []$ **do**
  $C \leftarrow head(wl)$
  $wl \leftarrow tail(wl)$
  if $\neg\exists C' \in Closed\ s.t. C' \preccurlyeq C$ **then**
    foreach $i \in supp(C) \cup \{1 + \max supp(c)\}$ **do**
      foreach $\sigma \in \Sigma$ **do**
        foreach $C' s.t. C \xrightarrow{\sigma:i} C'$ and $C' \notin N$ **do**
          $N \leftarrow N \cup \{C'\}$
          $E \leftarrow E \cup \{C \xrightarrow{\sigma:i} C'\}$
          if C is accepting **then**
            return a word w labeling a path in the graph (N, E) from C to a root
          else
            $wl \leftarrow wl \mathbin{++} [C']$
  $Closed \leftarrow Closed \cup \{C\}$
return Empty

Configurations and Coverings

- A Configuration, C, Accepts iff $\{q | q(i_0, \cdots, i_{ar(q)}) \in C\} \subseteq F$
- $C \overset{\sigma:k}{\longrightarrow} C'$ iff C' is a cube of (in DNF)

$$\bigwedge_{q(i_1,\cdots,i_{ar(q)})\in C} \delta(q,\sigma)[\mathrm{i}_o \mapsto k, \mathrm{i}_1 \mapsto i_1, \cdots, \mathrm{i}_{ar(q)} \mapsto i_{ar(q)}]$$

- If $C \preccurlyeq C'$,

  If C' is accepting then C must be accepting

  If $C' \overset{\sigma:j}{\longrightarrow} \overline{C'}$ then $\exists k, C \overset{\delta:k}{\longrightarrow} \overline{C}$ and $\overline{C} \preccurlyeq \overline{C'}$

  Therefore, if C' can reach an accepting state then so must C

Covering Relation (≼)

$$C = \{q(1,2), q(1,3), r(2)\}$$

Covering Relation ($\preccurlyeq$)

$$C = \{q(1,2), q(1,3), r(2)\}$$

$$supp(C) = \{1,2,3\}$$

Covering Relation (≼)

$C = \{q(1,2), q(1,3), r(2)\}$ $\quad$ $C' = \{q(8,7), q(8,6), r(7), r(6)\}$

$supp(C) = \{1,2,3\}$

Covering Relation ($\preccurlyeq$)

$C = \{q(1,2), q(1,3), r(2)\}$

$supp(C) = \{1,2,3\}$

$C' = \{q(8,7), q(8,6), r(7), r(6)\}$

$supp(C') = \{6,7,8\}$

Covering Relation (≼)

$C = \{q(1,2), q(1,3), r(2)\}$

$supp(C) = \{1,2,3\}$

$C' = \{q(8,7), q(8,6), r(7), r(6)\}$

$supp(C') = \{6,7,8\}$

$$C \preccurlyeq C'$$

Covering Relation ($\preccurlyeq$)

$$C = \{q(1,2), q(1,3), r(2)\}$$

$$supp(C) = \{1,2,3\}$$

$$C' = \{q(8,7), q(8,6), r(7), r(6)\}$$

$$supp(C') = \{6,7,8\}$$

$$C \preccurlyeq C'$$

$$\pi = \{1 \mapsto 8, 2 \mapsto 6, 3 \mapsto 7, \dots\}$$

Covering Relation (≼)

$$C = \{q(8,6), q(8,7), r(6)\} \quad \subseteq \quad C' = \{q(8,7), q(8,6), r(7), r(6)\}$$

$$supp(C) = \{1,2,3\} \qquad supp(C') = \{6,7,8\}$$

$$C \preccurlyeq C'$$

$$\pi = \{1 \mapsto 8, 2 \mapsto 6, 3 \mapsto 7, \dots\}$$

Covering Relation ($\preccurlyeq$)

$$C = \{q(8,7), q(8,6), r(7)\} \quad \subseteq \quad C' = \{q(8,7), q(8,6), r(7), r(6)\}$$

$$supp(C) = \{1,2,3\} \qquad\qquad supp(C') = \{6,7,8\}$$

$$C \preccurlyeq C'$$

$$\pi = \{1 \mapsto 8, 2 \mapsto 6, 3 \mapsto 7, \dots\} \qquad \pi = \{1 \mapsto 8, 2 \mapsto 7, 3 \mapsto 6, \dots\}$$

Covering Relation (≼)

$$C = \{q(0), r(1)\}$$

Covering Relation ($\preccurlyeq$)

$$C = \{q(0), r(1)\}$$

$$supp(C) = \{0,1\}$$

Covering Relation ($\preccurlyeq$)

$C = \{q(0), r(1)\}$

$supp(C) = \{0,1\}$

$C' = \{q(2), r(2)\}$

Covering Relation (≼)

$C = \{q(0), r(1)\}$

$supp(C) = \{0,1\}$

$C' = \{q(2), r(2)\}$

$supp(C') = \{2\}$

Covering Relation ($\preccurlyeq$)

$C = \{q(0), r(1)\}$

$supp(C) = \{0,1\}$

$C' = \{q(2), r(2)\}$

$supp(C') = \{2\}$

$$C \not\preccurlyeq C'$$

Covering Relation (≼)

$C = \{q(0), r(1)\}$

$supp(C) = \{0,1\}$

$C' = \{q(2), r(2)\}$

$supp(C') = \{2\}$

$C \not\preccurlyeq C'$

π must be a permutation (injective)

Covering Relation (≼)

- For configurations C and C’, C **covers** C’ ($C \preccurlyeq C'$)

Covering Relation ($\preccurlyeq$)

- For configurations C and C’, C **covers** C’ ($C \preccurlyeq C'$)

  $\exists\, \pi : \mathbb{N} \to \mathbb{N}, \forall\, q \in Q,$

  $$q(i_1, \cdots, i_{ar(q)}) \in C \to q\left(\pi(i_1), \cdots, \pi(i_{ar(q)})\right) \in C'$$

Covering Relation ($\preccurlyeq$)

- For configurations C and C', C **covers** C' ($C \preccurlyeq C'$)

  $\exists\, \pi : \mathbb{N} \to \mathbb{N}, \forall\, q \in Q,$

  $$q\left(i_1, \cdots, i_{ar(q)}\right) \in C \to q\left(\pi(i_1), \cdots, \pi\left(i_{ar(q)}\right)\right) \in C'$$

  Alternatively,

  $$\left\{q\left(\pi(i_1), \cdots, \pi\left(i_{ar(q)}\right)\right) \middle| q\left(i_1, \cdots, i_{ar(q)}\right) \in C\right\} \subseteq C'$$

Covering Relation ($\preccurlyeq$)

- For configurations C and C', C **covers** C' ($C \preccurlyeq C'$)

  $\exists \pi : \mathbb{N} \to \mathbb{N}, \forall q \in Q,$

  $$q(i_1, \cdots, i_{ar(q)}) \in C \to q\left(\pi(i_1), \cdots, \pi(i_{ar(q)})\right) \in C'$$

  Alternatively,

  $$\left\{q\left(\pi(i_1), \cdots, \pi(i_{ar(q)})\right) \middle| q(i_1, \cdots, i_{ar(q)}) \in C\right\} \subseteq C'$$

- Downward Compatibility with PA[1,2]

[Kincaid et. al. 2015][1] [Finkel and Schnoebelen. 2001][2]