Pointer Programs and Undirected Reachability

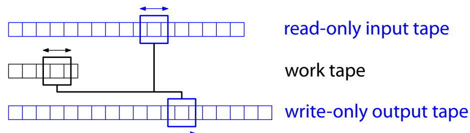
Martin Hofmann Ulrich Schöpp

Ludwig-Maximilians-Universität München

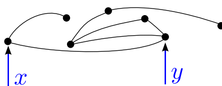
LICS 2009

Pointer Programs and Logarithmic Space

The complexity class LOGSPACE is defined by Turing Machines with logarithmic space usage.



In practice, LOGSPACE algorithms are usually intended to operate on structured data.



Inputs are accessed by abstract operations, bit-level encoding details are not important.

Pure Pointer Programs

Many LOGSPACE algorithms are presented as pointer programs that do not inspect or manipulate bit-level encodings.

Pure Pointer Programs

- read-only input, accessed by a constant number of pointers
- abstract pointers without internal structure
- no manipulation of large unstructured data
  (e.g. tapes of a Turing Machine)

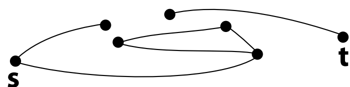
How good are pure pointer programs as an intuition for LOGSPACE?

Are there natural LOGSPACE problems that cannot be expressed as pure pointer programs?

Where does one need bit-level encodings or unstructured data?

Undirected **s**-**t**-Reachability

USTCON — Are **s** and **t** connected by an undirected path?



[Reingold 2005] shows that this problem is in LOGSPACE.

Reingold's algorithm is not a pure pointer program — it uses counting registers of logarithmic size in addition to pointers.

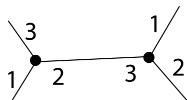
We show that there is no pure pointer program for USTCON.

Pure Pointer Language (PURPLE)

Simple `while`-language with `forall`-iteration:

$$\texttt{skip} \mid M; M \mid x^\Gamma := t^\Gamma \mid x^{\texttt{bool}} := t^{\texttt{bool}} \mid \texttt{if } t^{\texttt{bool}} \texttt{ then } M \texttt{ else } M$$
$$\mid \texttt{while } t^{\texttt{bool}} \texttt{ do } M \mid \texttt{forall } x^\Gamma \texttt{ do } M$$

Terms for graph nodes and booleans:



$$t^\Gamma \quad ::= \quad x^\Gamma \mid \mathbf{s} \mid \mathbf{t} \mid t^\Gamma.\mathit{succ}(i)$$
$$t^{\texttt{bool}} \quad ::= \quad x^{\texttt{bool}} \mid t^\Gamma = t^\Gamma \mid \text{usual boolean combinations}$$

The iteration order in the `forall`-loops is unspecified: To decide a problem, a program must give the right answer regardless of the order in which the nodes are presented in the `forall`-loops.

Iteration with Unspecified Order

java.util

Class HashSet

```
java.lang.Object
  └ java.util.AbstractCollection
      └ java.util.AbstractSet
          └ java.util.HashSet
```

All Implemented Interfaces:

Cloneable, Collection, Serializable, Set

Direct Known Subclasses:

JobStateReasons, LinkedHashSet

---

public class **HashSet**
extends AbstractSet
implements Set, Cloneable, Serializable

This class implements the `Set` interface, backed by a hash table (actually a `HashMap` instance). **It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time.** This class permits the `null` element.

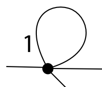
Iteration in PURPLE

Examples

- Parity

$$\textit{even} := \texttt{true};$$
$$\texttt{forall}\ x\ \texttt{do}\ \textit{even} := \neg \textit{even}$$

- Checking for the existence of a node with a loop



$$\textit{hasloop} := \texttt{false};$$
$$\texttt{forall}\ x\ \texttt{do}\ \textit{hasloop} := \textit{hasloop} \lor (x = x.\textit{succ}(1))$$

Iteration vs. Total Ordering

- total ordering can be used for iteration
- total ordering also allows one to encode arbitrary data in graph nodes
- unspecified order in `forall`-loops prevents unwanted encoding of arbitrary data [H. & Sch. 2008]

Other Formalisms

PURPLE subsumes other formalisms for programming with abstract pointers.

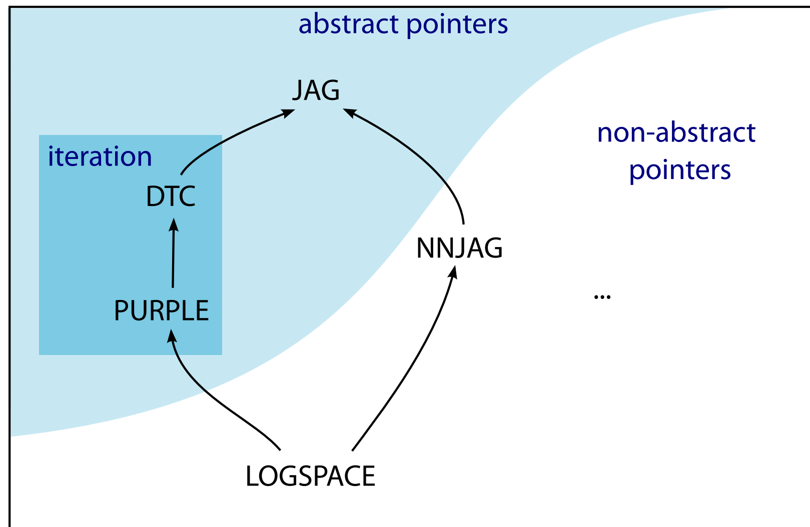
Jumping Automata on Graphs (JAGs) [Cook & Rackoff 1980]

- automata version of `forall`-free PURPLE programs
- JAGs cannot reach isolated components of the input graph

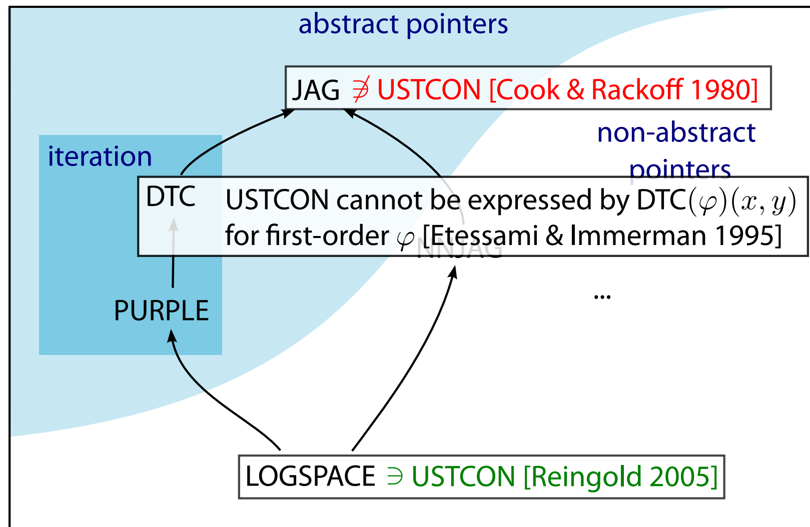
Deterministic Transitive Closure (DTC) logic for locally ordered graphs [Etessami & Immerman 1995].

- first-order logic with deterministic transitive closure, variables range over graph nodes
- PURPLE can evaluate DTC-formulae
- PURPLE refined quantification $\forall, \exists$ into iteration
- DTC-logic cannot express parity

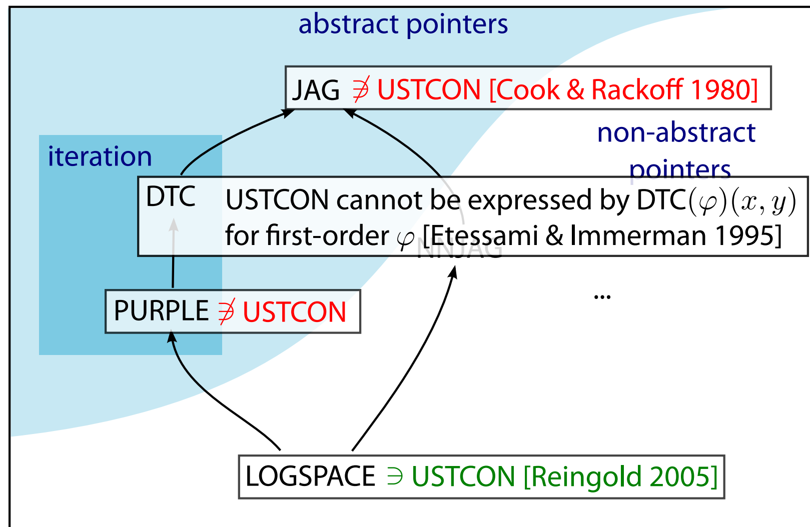
Pointer Programs and Undirected Reachability



Pointer Programs and Undirected Reachability



Pointer Programs and Undirected Reachability



Pointer Programs and Undirected Reachability

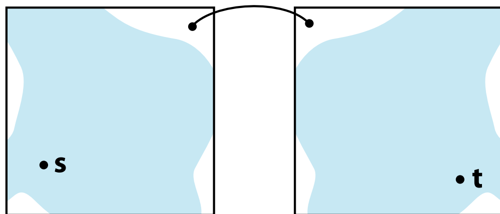
Theorem There is no PURPLE program that decides **s**-**t**-reachability in undirected graphs of degree 3.

Corollary There is no DTC-formula for locally ordered graphs that decides **s**-**t**-reachability in undirected graphs of degree 3.

Without `forall`-loops

Theorem [Cook & Rackoff 1980]
There is no `forall`-free PURPLE program that decides **s**-**t**-reachability in undirected graphs of degree 3.

- Suppose there exists such a program.
- Construct graph in which program is confined to the blue part (can assume all variables to be on **s** or **t** initially).

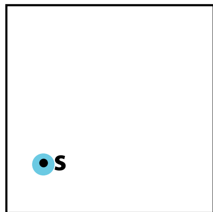


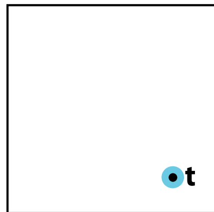
- Purported program gives the same result if we remove the edge between the two components.

Locality of `forall`-free programs

Cook & Rackoff's proof does not generalise to PURPLE programs with `forall` loops.

We use a generalisation of Cook & Rackoff's method to show that `forall`-free programs are confined to very small areas of certain graphs.

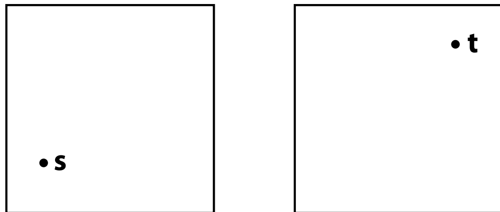




PURPLE cannot decide Undirected Reachability

Assume a PURPLE program M decides USTCON.

It suffices to construct a graph of the form



that is accepted by one run of M.

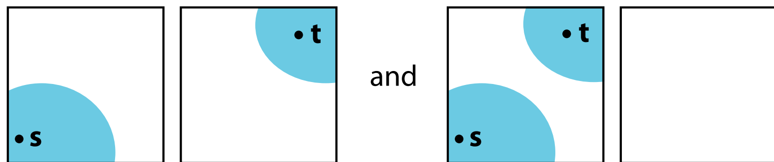
PURPLE cannot decide Undirected Reachability

Show that M can be implemented by a loop-free program N on a specially constructed graph.

From any start configuration, N reaches an end configuration that can also be reached by one run of M.

Graph has homogeneous structure and diameter larger than twice the range of N.

$\Rightarrow$ N cannot distinguish between



$\Rightarrow$ One run of M accepts the left graph ↯

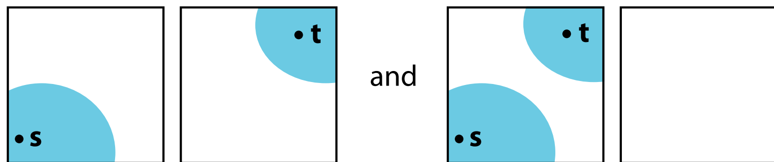
PURPLE cannot decide Undirected Reachability

Show that M can be implemented by a loop-free program N on a specially constructed graph.

From any start configuration, N reaches an end configuration that can also be reached by one run of M.

Graph has homogeneous structure and diameter larger than twice the range of N.

$\Rightarrow$ N cannot distinguish between



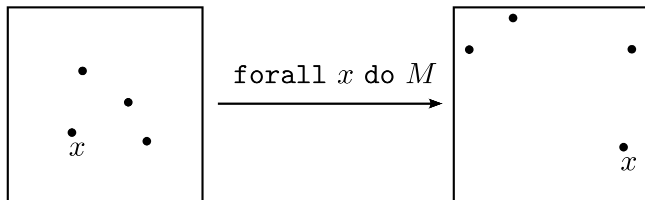
$\Rightarrow$ One run of M accepts the left graph ↯

Eliminating loops from PURPLE programs

On a very special class of graphs, (one run of) each PURPLE program can be implemented by a loop-free program.

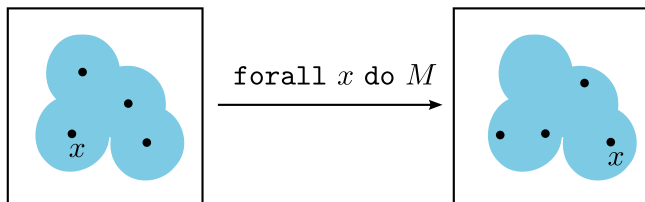
`while`-loops can be unfolded into nested `forall`-loops.

Elimination of `forall`-loops by induction on the program.



Eliminating loops from PURPLE programs

Choose the iteration order so that at the end all variables lie in a small neighbourhood of the original positions.



Simulate this step by a (huge) loop-free program that can fully explore the blue neighbourhood.

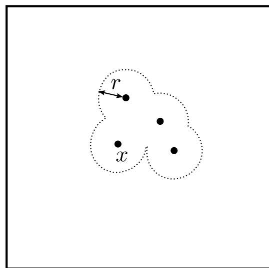
The graph is so large that after the elimination of all loops, its diameter is still more than twice the range of the resulting program.

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

There exists a (small enough) number r such that:
If in each iteration step we place x at least $2r$ away from all other variables, then at any time all variables are an r-neighbourhood of

- the original variable positions; or
- the first or last few choices for x.

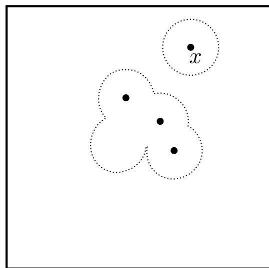


Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

There exists a (small enough) number r such that:
If in each iteration step we place x at least $2r$ away from all other variables, then at any time all variables are an r-neighbourhood of

- the original variable positions; or
- the first or last few choices for x.

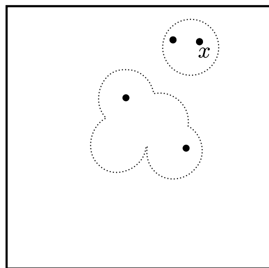


Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

There exists a (small enough) number r such that:
If in each iteration step we place x at least $2r$ away from all other variables, then at any time all variables are an r-neighbourhood of

- the original variable positions; or
- the first or last few choices for x.

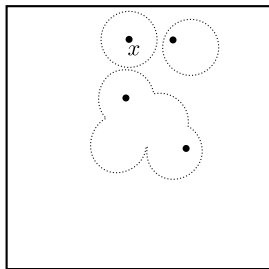


Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

There exists a (small enough) number r such that:
If in each iteration step we place x at least $2r$ away from all other variables, then at any time all variables are an r-neighbourhood of

- the original variable positions; or
- the first or last few choices for x.

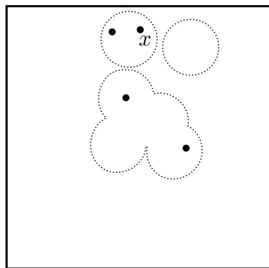


Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

There exists a (small enough) number r such that:
If in each iteration step we place x at least $2r$ away from all other variables, then at any time all variables are an r-neighbourhood of

- the original variable positions; or
- the first or last few choices for x.

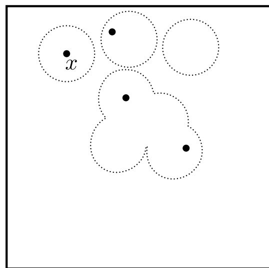


Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

There exists a (small enough) number r such that:
If in each iteration step we place x at least $2r$ away from all other variables, then at any time all variables are an r-neighbourhood of

- the original variable positions; or
- the first or last few choices for x.

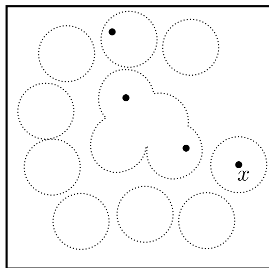


Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

There exists a (small enough) number r such that:
If in each iteration step we place x at least $2r$ away from all other variables, then at any time all variables are an r-neighbourhood of

- the original variable positions; or
- the first or last few choices for x.

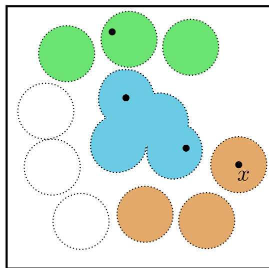


Choosing the Iteration Order for `forall` x `do` M

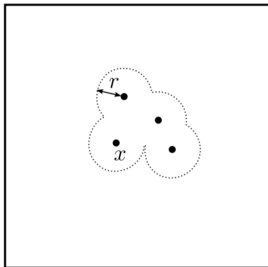
1. How do the variable positions depend on the iteration order?

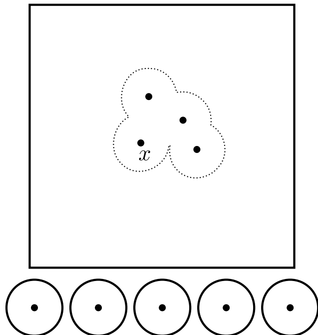
There exists a (small enough) number r such that:
If in each iteration step we place x at least $2r$ away from all other variables, then at any time all variables are an r-neighbourhood of

- the original variable positions; or
- the first or last few choices for x.

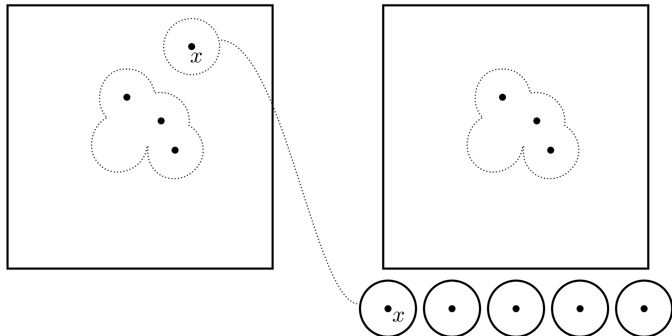


Choosing the Iteration Order for `forall` x `do` M

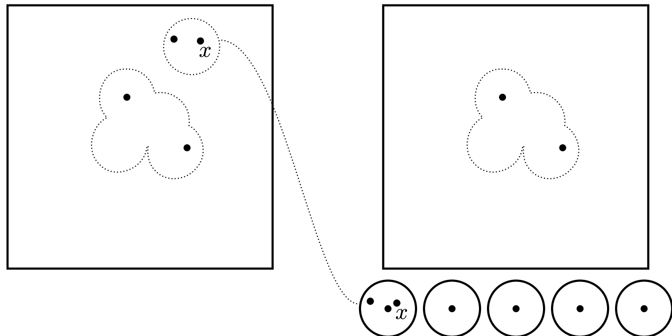




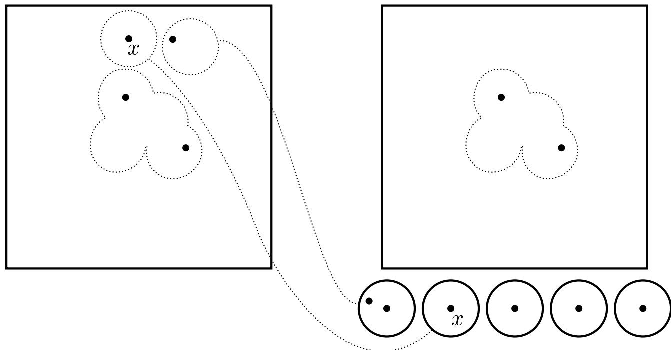
Choosing the Iteration Order for `forall` x `do` M



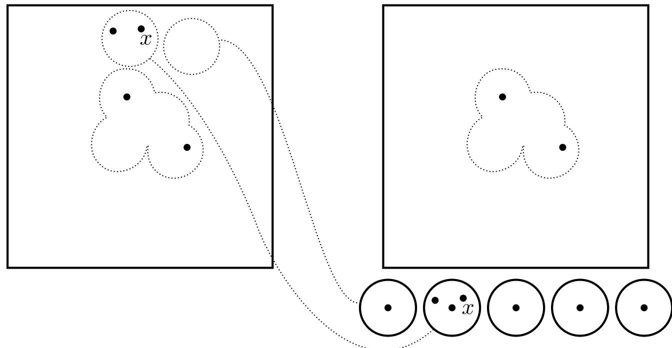
Choosing the Iteration Order for `forall` x `do` M



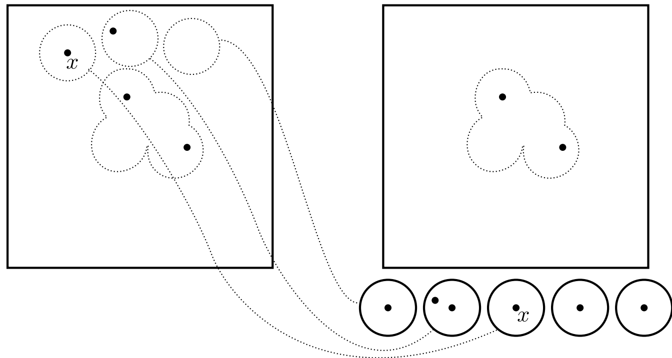
Choosing the Iteration Order for `forall` x `do` M



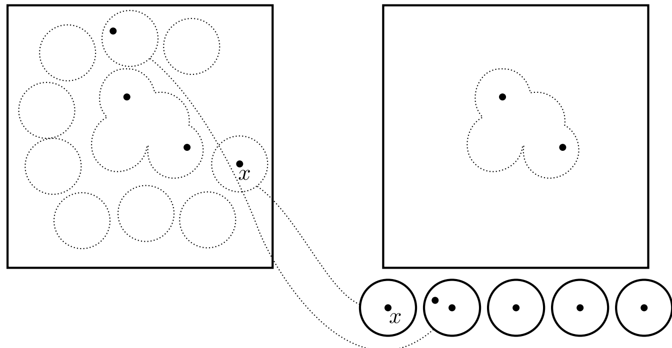
Choosing the Iteration Order for `forall` x `do` M



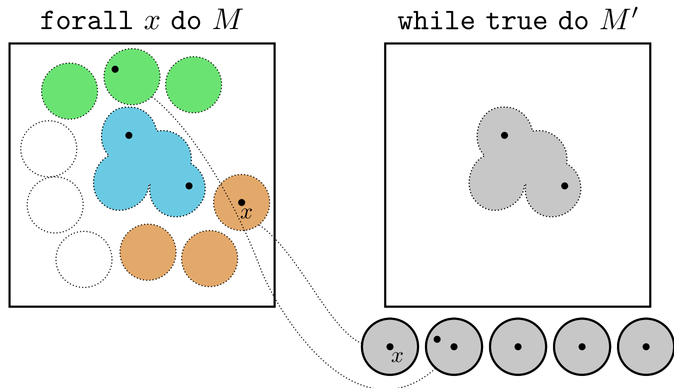
Choosing the Iteration Order for `forall` x `do` M



Choosing the Iteration Order for $\texttt{forall}\ x\ \texttt{do}\ M$



Choosing the Iteration Order for `forall` x `do` M



- Range of `while true do` M' can be bounded by small r.
  (by graph construction and a generalisation of the argument of [Cook & Rackoff 1980] — formalised in Coq [Sch., LPAR 2008])

$\Rightarrow$ Moves of both programs repeat periodically.

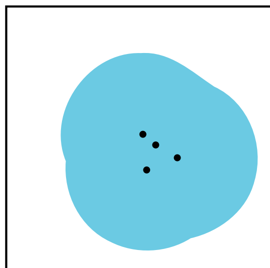
$\Rightarrow$ Variables must be r-close to first or last few jump destinations.

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



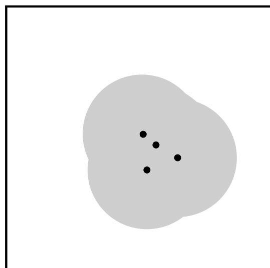
Find an enumeration that ends with all variables in the blue area.

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



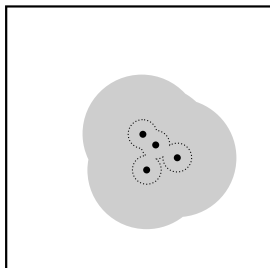
Start with a smaller area around the initial positions and iterate as follows:

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



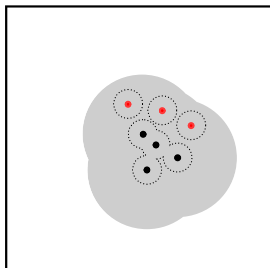
Start with a smaller area around the initial positions and iterate as follows:

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



Start with a smaller area around the initial positions and iterate as follows:

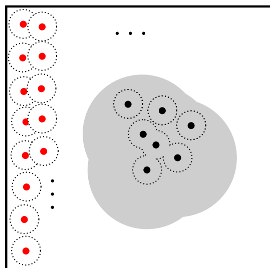
1. First few destinations in vicinity

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



Start with a smaller area around the initial positions and iterate as follows:

1. First few destinations in vicinity
2. Enumerate all nodes outside the grey area

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.

Start with a smaller area around the initial positions and iterate as follows:

1. First few destinations in vicinity
2. Enumerate all nodes outside the grey area

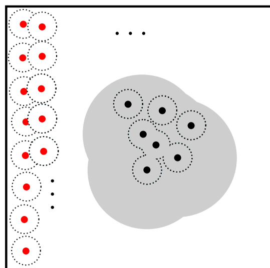
   Use sightseer enumeration:
   At any time the last few nodes are far appart.

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



Start with a smaller area around the initial positions and iterate as follows:

1. First few destinations in vicinity
2. Enumerate all nodes outside the grey area

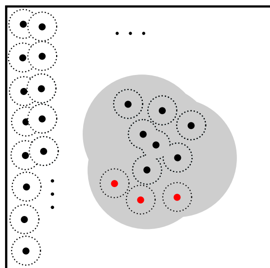
   Use sightseer enumeration:
   At any time the last few nodes are far appart.

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



Start with a smaller area around the initial positions and iterate as follows:

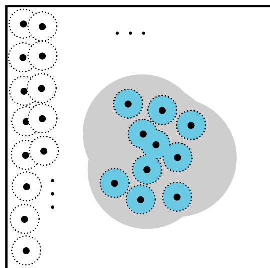
1. First few destinations in vicinity
2. Enumerate all nodes outside the grey area
3. Next few destinations in vicinity

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



Start with a smaller area around the initial positions and iterate as follows:

1. First few destinations in vicinity
2. Enumerate all nodes outside the grey area
3. Next few destinations in vicinity

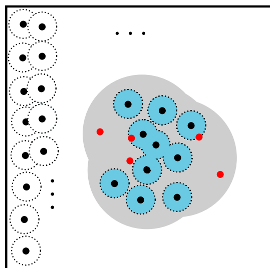
No variable is left outside grey area!

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



Start with a smaller area around the initial positions and iterate as follows:

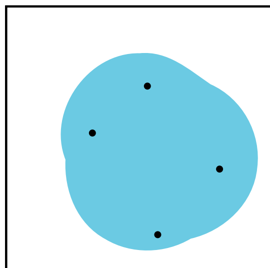
1. First few destinations in vicinity
2. Enumerate all nodes outside the grey area
3. Next few destinations in vicinity
4. Remaining nodes (from grey area)

Choosing the Iteration Order for `forall` x `do` M

1. How do the variable positions depend on the iteration order?

As long as all iteration jumps are $2r$ away from all other variables, the final variable positions will be r-close to original positions or the first/last few jump destinations.

2. Choose a graph enumeration so that at the end all variables are close to the original positions.



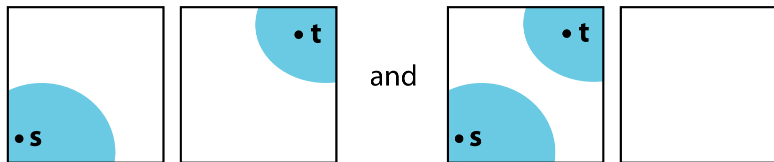
Start with a smaller area around the initial positions and iterate as follows:

1. First few destinations in vicinity
2. Enumerate all nodes outside the grey area
3. Next few destinations in vicinity
4. Remaining nodes (from grey area)

Eliminating loops from PURPLE programs

On a very special class of graphs, (one run of) each PURPLE program can be implemented by a loop-free program.

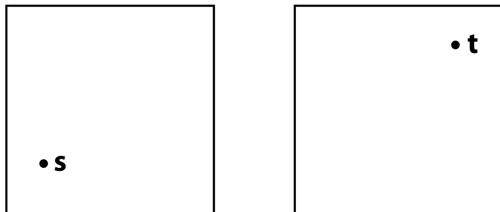
The loop-free program has limited range and therefore cannot distinguish between



$\Rightarrow$ PURPLE cannot decide **s**-**t**-reachability in undirected graphs.

Graph Construction

The graph



consists of two disjoint copies of a Cayley Graph.

- Nodes are the elements of a group G.
- Edges are determined by a generating set S of the group.
  For all $g \in G$ and $s \in S$ there is an edge from g to $g \cdot s$.

Graph Construction

We use a group with

- a huge set of elements,
- a small set of generators,
- a small exponent.

The exponent of G is the smallest number $\exp(G)$ such that

$$\forall g \in G.\ \ g^{\exp(G)} = e \ \ .$$

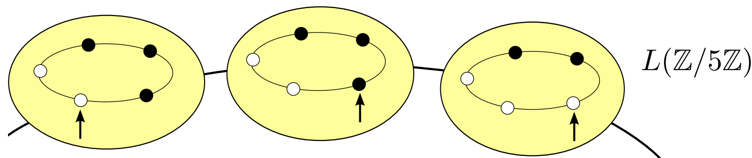
The exponent determines how quickly a `forall`-free program starts running around in circles.

Iterated Wreath Product / Lamplighter Construction

We construct such a group by iterating the wreath product $L(G) = (\mathbb{Z}/2\mathbb{Z}) \wr G$, starting with $\mathbb{Z}/m\mathbb{Z}$.

The Cayley graphs thus obtained may be described in terms of the lamplighter construction.

- Cayley graph of G — layout of street lamps
- Cayley graph of $L(G)$ — lighting of lamps by a lamplighter
  - Nodes: lamplighter position and lamp states ($G \times 2^G$)
  - Edges for switching on/off a lamp and for moving to a neighbouring lamp



Iterated Wreath Product / Lamplighter Construction

Properties of $L^i(\mathbb{Z}/m\mathbb{Z})$:

- Number of nodes: $\exp^i(m) = 2^{2^{\cdot^{\cdot^{\cdot^{2^m}}}}}$ (tower of height i)
- Number of generators: $i + 2$
- Exponent: $m \cdot 2^i$

Iterated Wreath Product / Lamplighter Construction

Properties of $L^i(\mathbb{Z}/m\mathbb{Z})$:

- Number of nodes: $\exp^i(m) = 2^{2^{\cdot^{\cdot^{\cdot^{2^m}}}}}$ (tower of height i)
- Number of generators: $i + 2$
- Exponent: $m \cdot 2^i$

Elimination of `forall`-loops from PURPLE program M results in a loop-free program whose range is tower of constant height.

$$\exp^r(i + m) = 2^{2^{\cdot^{\cdot^{\cdot^{2^{i+m}}}}}}$$

(height depends on nesting depth of `forall`-loops, but not m or i)

Iterated Wreath Product / Lamplighter Construction

Properties of $L^i(\mathbb{Z}/m\mathbb{Z})$:

- Number of nodes: $\exp^i(m) = 2^{2^{\cdot^{\cdot^{\cdot^{2^m}}}}}$ (tower of height i)
- Number of generators: $i + 2$
- Exponent: $m \cdot 2^i$

Elimination of `forall`-loops from PURPLE program M results in a loop-free program whose range is tower of constant height.

$$\exp^r(i + m) = 2^{2^{\cdot^{\cdot^{\cdot^{2^{i+m}}}}}}$$

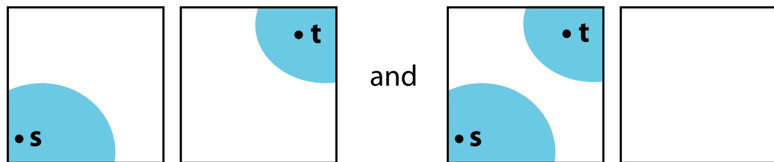
(height depends on nesting depth of `forall`-loops, but not m or i)

If we increase i, then the size of the graph grows much faster than the range of the `forall`-free program!

Eliminating loops from PURPLE programs

For each PURPLE program M we can choose i and m large enough so that M can be implemented on two disjoint copies of $L^i(m)$ by a loop-free program of small range.

The loop-free program cannot distinguish between



Theorem For all k there is a number d such that no `while`-free PURPLE program with `forall`-depth k decides reachability on undirected graphs of degree d.

Corollary There is no PURPLE program that decides **s**-**t**-reachability in undirected graphs of degree 3.

Conclusion

Undirected **s-t** reachability cannot be programmed with a constant number of abstract pointers.

- Analysis of popular programming methodology:
  Using iterators to traverse large data structures
- Programming Language/Automata methods used to answer a logical question about the expressivity of DTC-logic
- Iterated Wreath Products/Lamplighter Graphs, use of exponent as a graph parameter