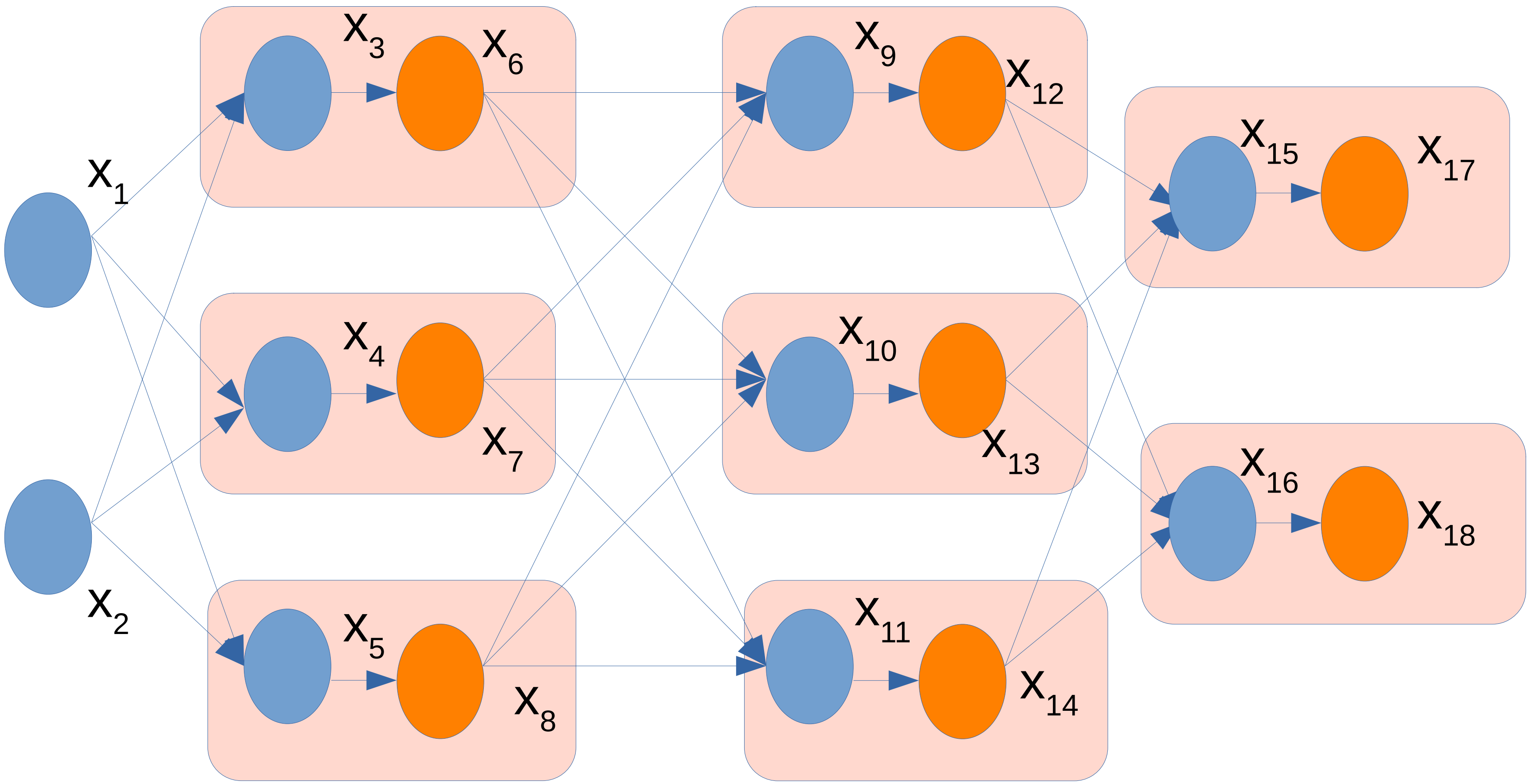
CS781: A Quick Primer on Abstract Interpretation for Neural Networks

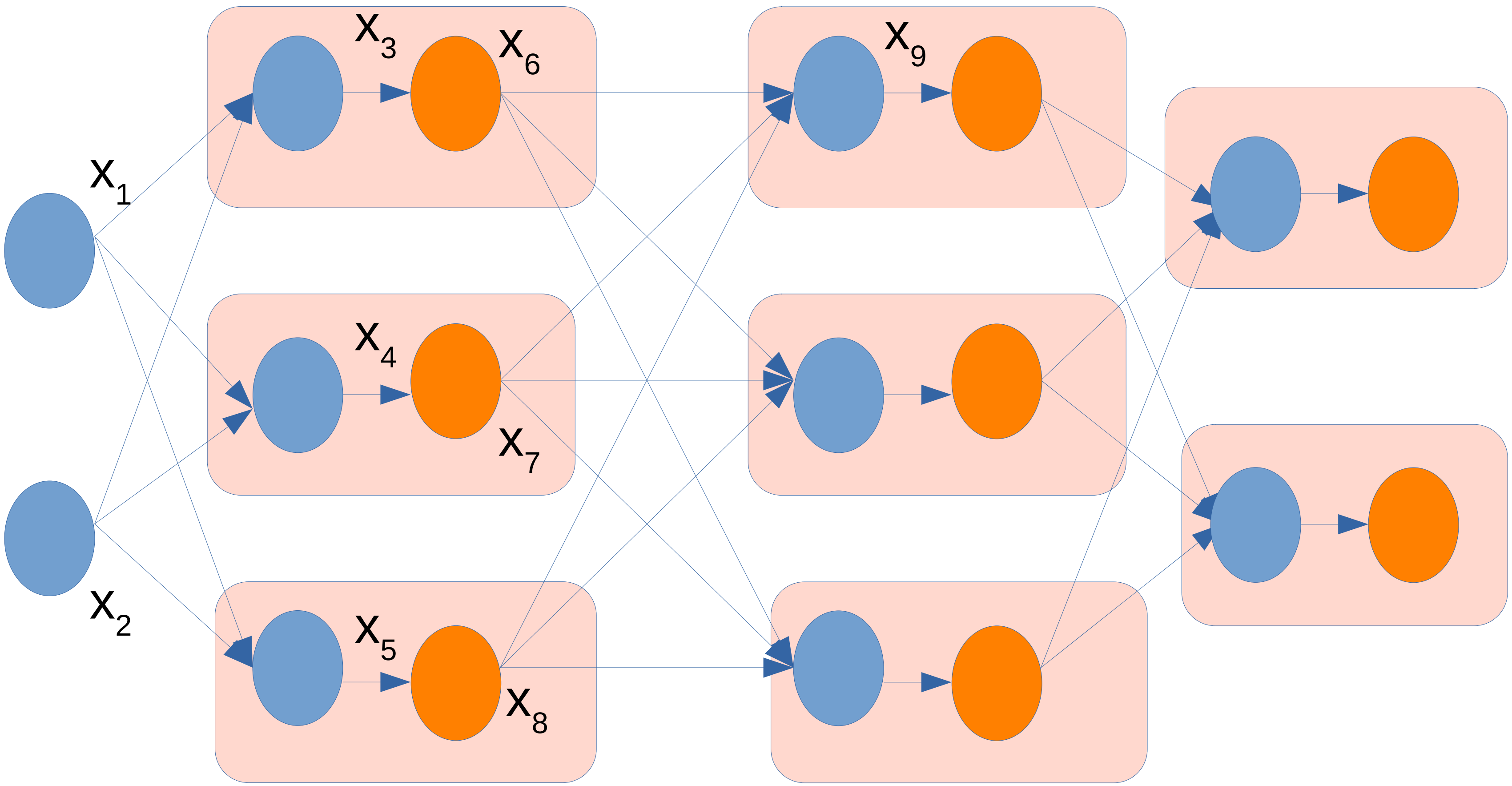
Supratik Chakraborty
IIT Bombay

Notion of State in Neural Network



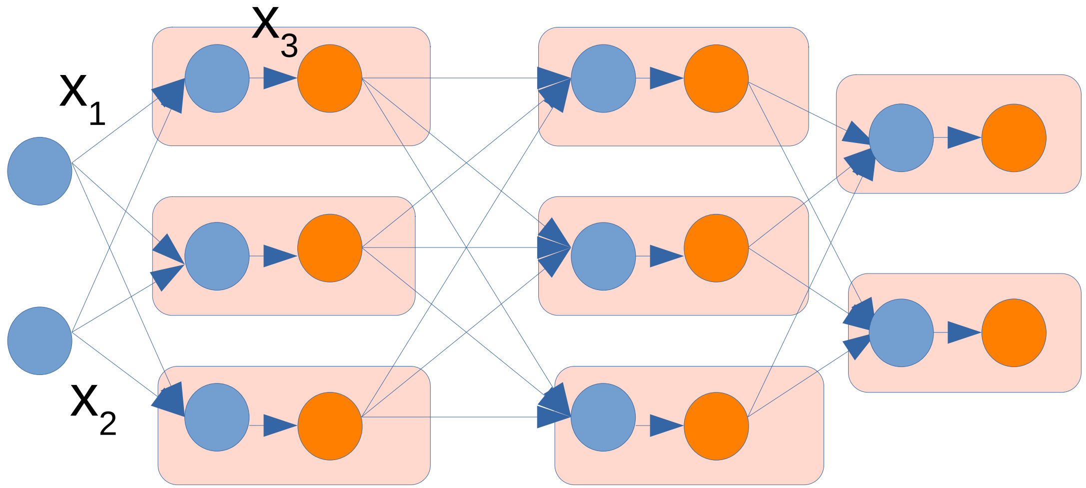
State: $(x_1, x_2, \ldots x_{18})$ in R^{18}

State Change in Feed-Forward Neural Network



$$(x'_1, x'_2, \ldots x'_{i-1}, x'_i) = f_i(x_1, x_2, \ldots x_{i-1}), \text{ for } i \text{ in } \{3, \ldots, 18\}$$

State Change in Feed-Forward NN as a sequence of instrns



$(x'_1, x'_2, x'_3) = f_3(x_1, x_2);$

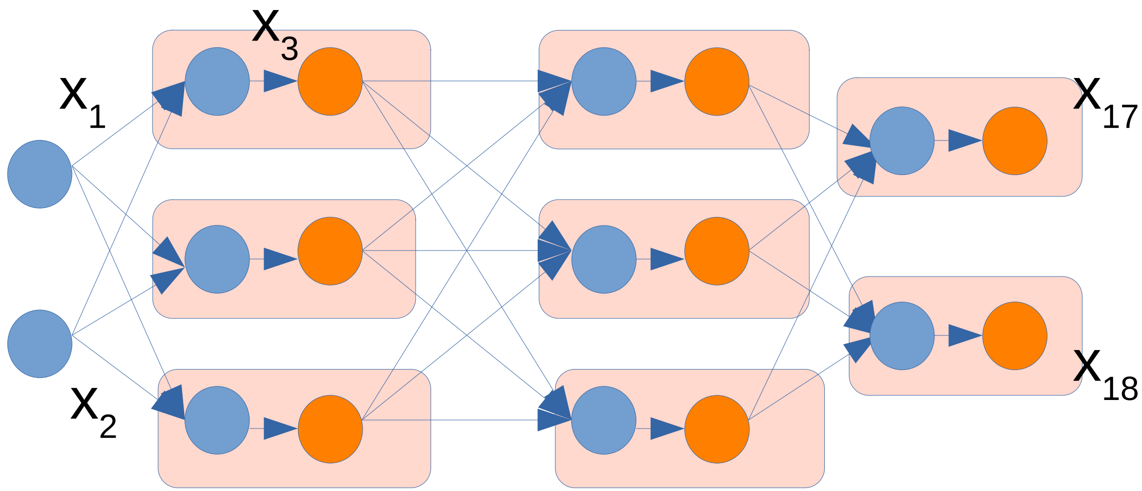
$(x''_1, x''_2, x''_3, x''_4) = f_4(x'_1, x'_2, x'_3);$

...

NN computation: a sequence of state transitions caused by seq of instructions

Proving Property of a FF NN

Pre-condition on (x1, x2)



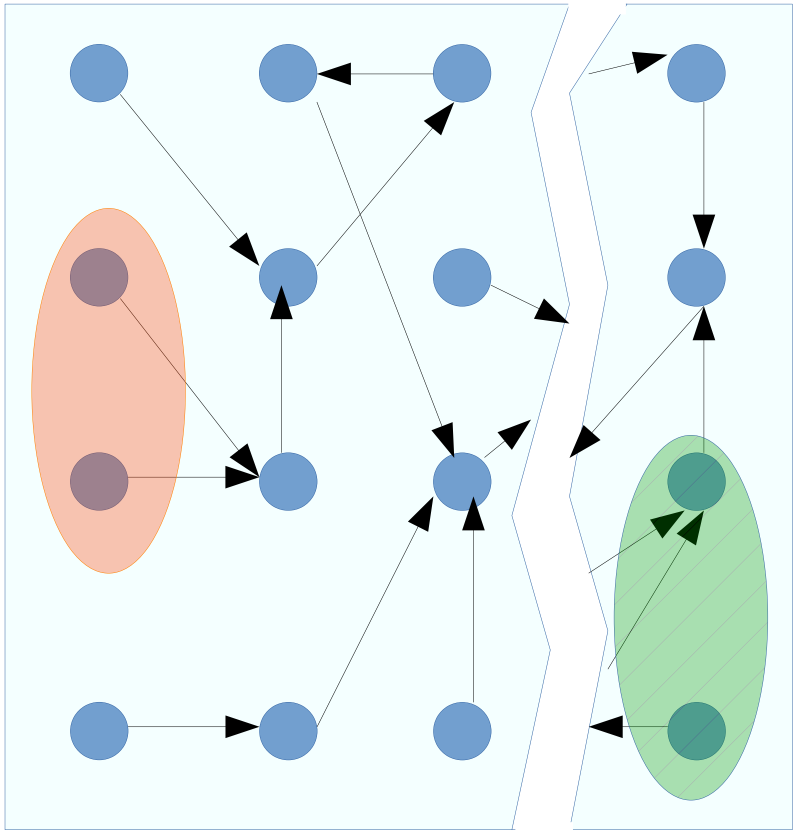
Post-condition on (x17, x18)

{Pre-condition on (x1, x2)}

$$(x'_1, x'_2, x'_3) = f_3(x_1, x_2);$$
$$(x''_1, x''_2, x''_3, x''_4) = f_4(x'_1, x'_2, x'_3);$$
$$\ldots$$

{Post-condition on (x17, x18) }

NN Computation as a State Transition System



{Pre-condition on (x1, x2)}

$$(x'_1, x'_2, x'_3) = f_3(x_1, x_2);$$
$$(x''_1, x''_2, x''_3, x''_4) = f_4(x'_1, x'_2, x'_3);$$
$$\ldots$$

{Post-condition on (x17, x18) }

Dealing with State Space Size

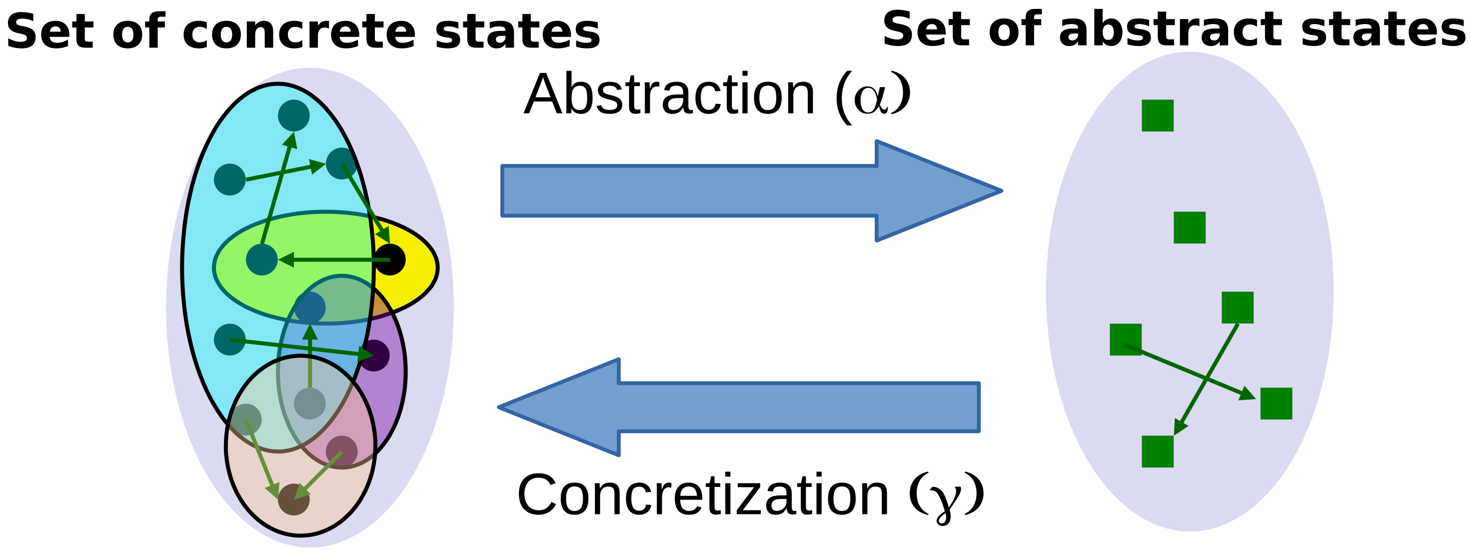
- Infinite state space
  - Difficult to represent using state transition diagram
  - Can we still do some reasoning?

Concrete states

- Solution: Use of abstraction
  - Naive view
    - Bunch sets of states together “intelligently”
    - Don't talk of individual states, talk of a representation of a set of states
    - Transitions between state set representations
  - Granularity of reasoning shifted
  - Extremely powerful general technique
    - Allows reasoning about large/infinite state spaces

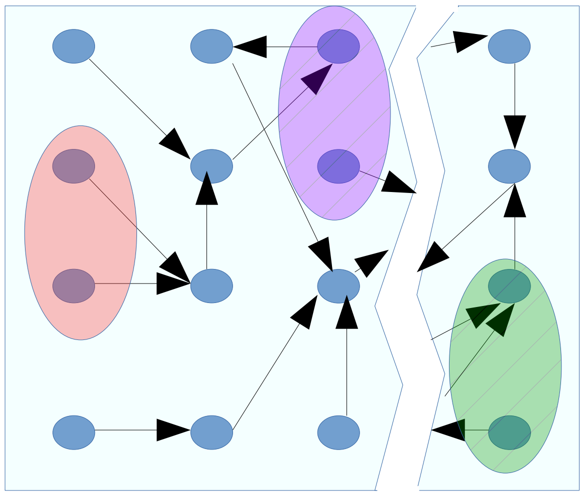
Abstract states

A Generic View of Abstraction



- Every subset of concrete states mapped to unique abstract state
- Desirable to capture containment relations
- Transitions between state sets (abstract states)

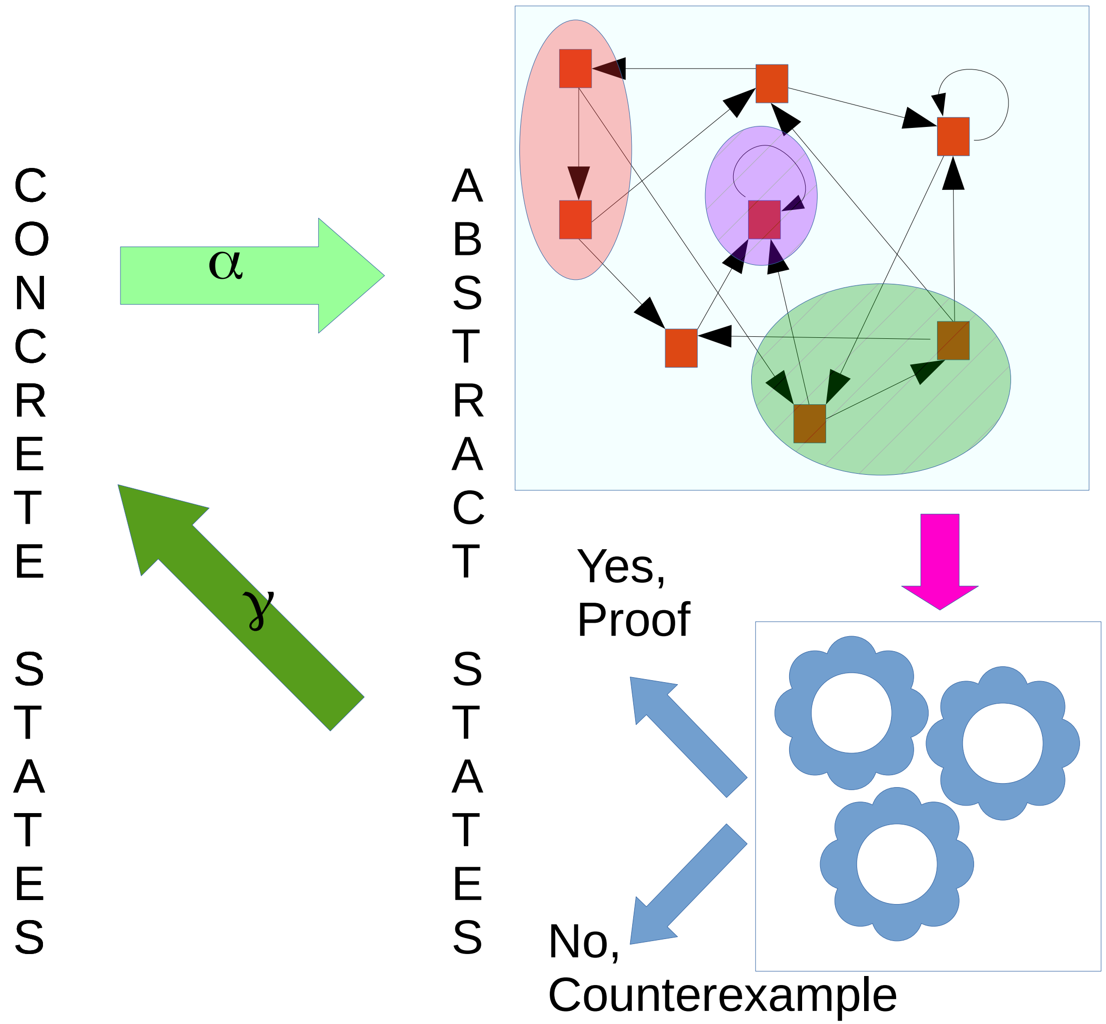
The Game Plan



Pre-condition:

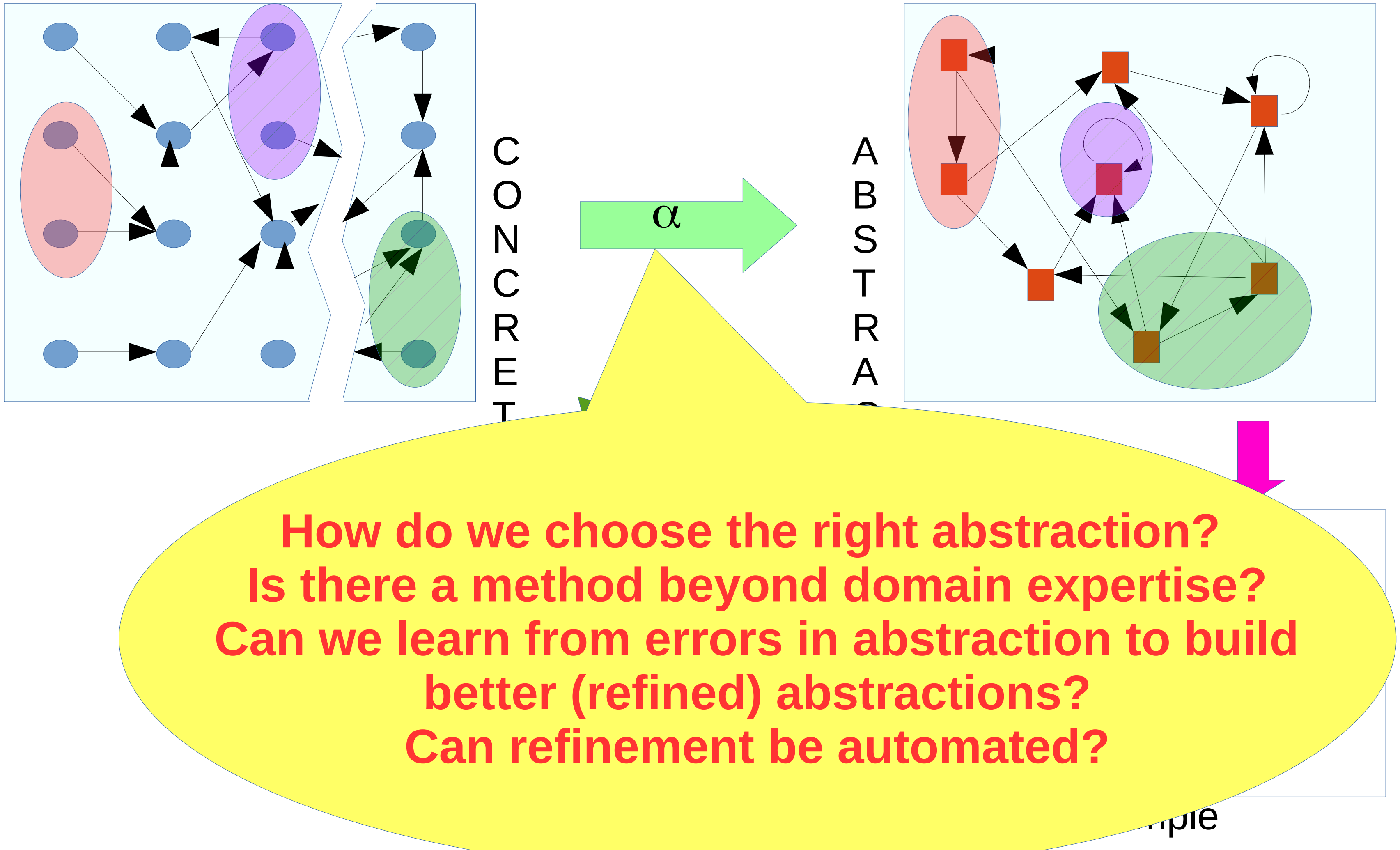
NN computation as a sequence of state transitions

Post-condition:



Abstract analysis engine

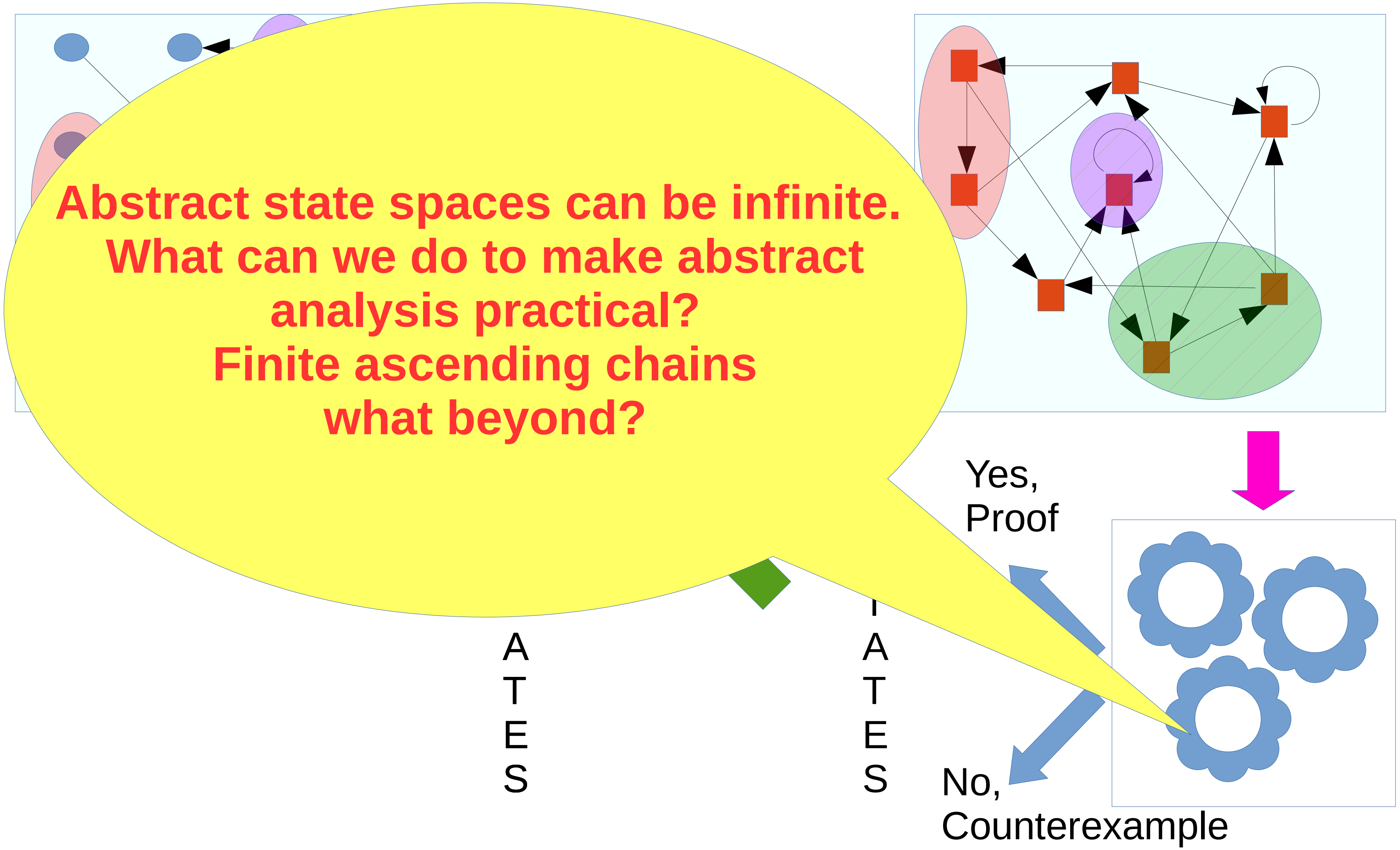
The Game Plan



How do we choose the right abstraction?
Is there a method beyond domain expertise?
Can we learn from errors in abstraction to build better (refined) abstractions?
Can refinement be automated?

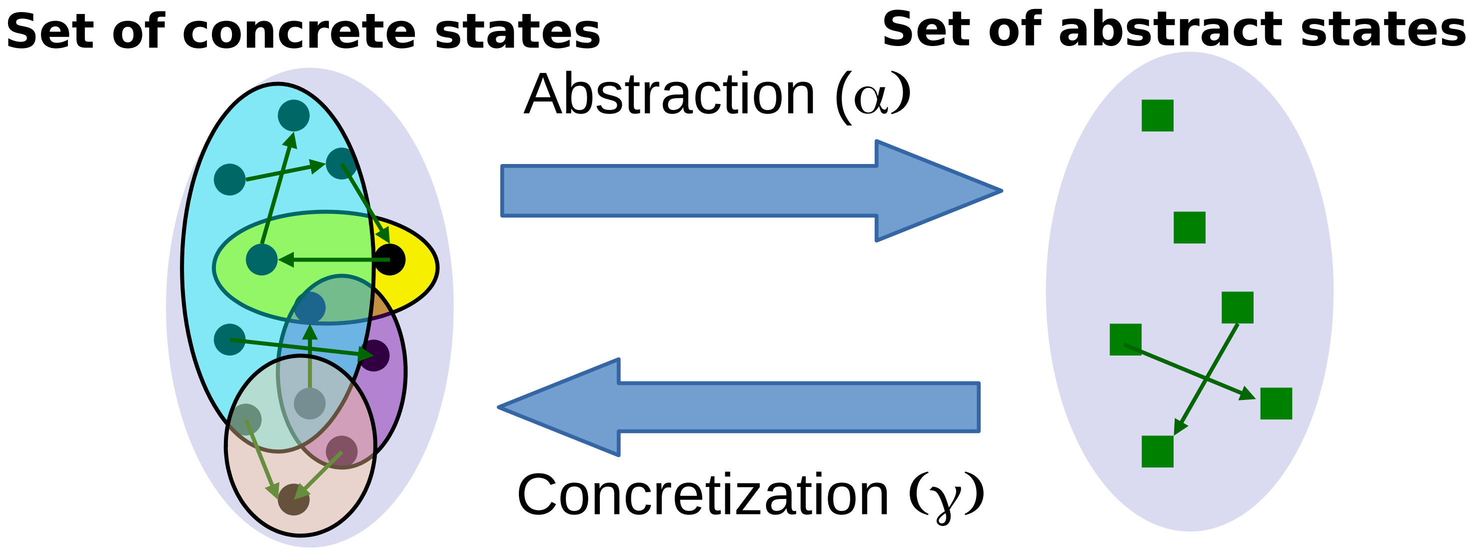
Abstract analysis engine

The Game Plan



Abstract analysis engine

Desirable Properties of Abstraction



- Suppose $S_1 \subseteq S_2$: subsets of concrete states
  - Any behaviour starting from S_1 can also happen starting from S_2
  - If $\alpha(S_1) = a_1, \alpha(S_2) = a_2$ we want this monotonicity in behaviour in abstr state space too
    - Need ordering of abstract states, similar in spirit to $S_1 \subseteq S_2$

Structure of Concrete State Space

- Set of concrete states: S
  - Concrete lattice C $= (\wp(S), \subseteq, \cup, \cap, S, \emptyset)$

Powerset of S

Partial order

Least upper bound

Greatest lower bound

Top element

Bottom element

Structure of Abstract State Space

- Abstract lattice $\mathsf{A} = (\mathcal{A}, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$
- Abstraction function $\alpha : \wp(S) \rightarrow \mathcal{A}$
  - Monotone: $S_1 \subseteq S_2 \Rightarrow \alpha(S_1) \sqsubseteq \alpha(S_2)$ for all $S_1, S_2 \subseteq S$
  - $\alpha(S) = \top, \quad \alpha(\emptyset) = \bot$
- Concretization function $\gamma : \mathcal{A} \rightarrow \wp(S)$
  - Monotone: $a_1 \sqsubseteq a_2 \Rightarrow \gamma(a_1) \subseteq \gamma(a_2)$ for all $a_1, a_2 \in \mathcal{A}$
  - $\gamma(\top) = S, \quad \gamma(\bot) = \emptyset$