ON DEPENDENT VARIABLES IN REACTIVE SYNTHESIS

Supratik Chakraborty

IIT Bombay

Joint work with S. Akshay, Eliyahu Basa and Dror Fried (TACAS 2024)

CAALM 2025, Paris

Systems with Input/Output Interaction

Input
signal from sensors

Output
actions made by AV

Systems with Input/Output Interaction

Input
signal from sensors

Output
actions made by AV

Specification
Don’t run into a wall

Systems with Input/Output Interaction

Input
signal from sensors

Output
actions made by AV

Specification

Don't run into a wall

Program/plan

- Stop when observes a wall nearby
- Slow down when observes a dead-end sign

Systems with Input/Output Interaction

Input
signal from sensors

Output
actions made by AV

Specification
Don’t run into a wall

Verification
- Testing
- Model Checking

Program/plan
- Stop when observes a wall nearby
- Slow down when observes a dead-end sign

Systems with Input/Output Interaction

Input
signal from sensors

Output
actions made by AV

Specification
Don’t run into a wall

Synthesis

Program/plan
- Stop when observes a wall nearby
- Slow down when observes a dead-end sign

Synthesis

Specification (declaration)

- Logical expression

→ Synthesis →

System (implementation)

- Program
- Circuit
- State Machine

Synthesis

Specification (declaration)

- Logical expression

Synthesis

System (implementation)

- Program
- Circuit
- State Machine

- First order logic - $\forall\, r, \forall r' \exists g \left(\left((r > r') \wedge Arrive(g, r') \right) \rightarrow Arrive(g, r) \right)$ UNDECIDABLE
- Temporal logic - ***always*** $(r \rightarrow (g \vee$ ***next*** $g))$ 2EXPTIME - COMPLETE
- Boolean logic - $\neg(r \wedge r') \rightarrow (r \vee g) \wedge (r' \vee g)$ in EXPTIME

Synthesis

Specification (declaration)

- Logical expression

Synthesis

System (implementation)

- Program
- Circuit
- State Machine

- First order logic - $\forall\, r, \forall r' \exists g \left(\left((r > r') \wedge Arrive(g, r')\right) \rightarrow Arrive(g, r)\right)$ UNDECIDABLE
- Temporal logic - ***always*** $(r \rightarrow (g \vee$ ***next*** $g))$ 2EXPTIME - COMPLETE
- Boolean logic - $\neg(r \wedge r') \rightarrow (r \vee g) \wedge (r' \vee g)$ in EXPTIME

Synthesis: From Specification to a Program

Functional systems

Input

Output

Synthesis: From Specification to a Program

Functional systems

Input

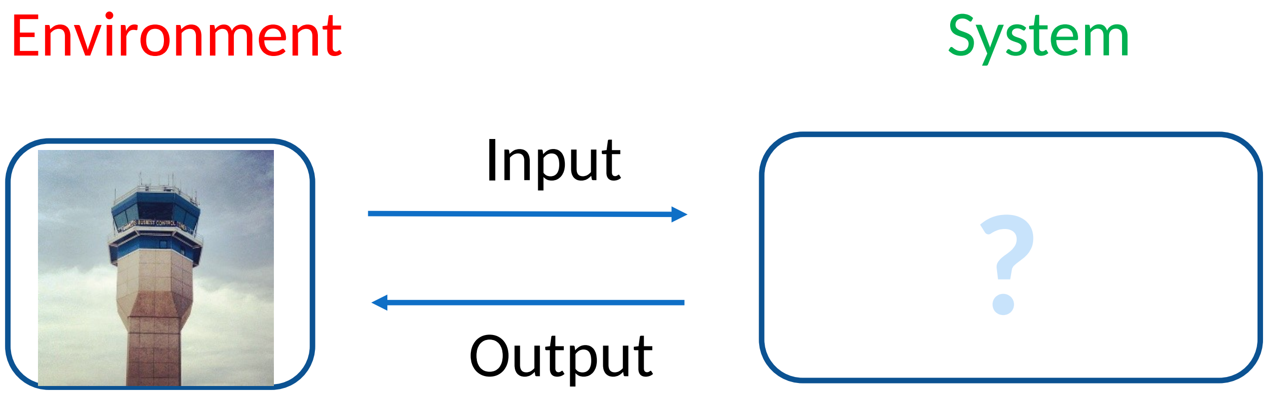
Output

Reactive systems

Input

Output

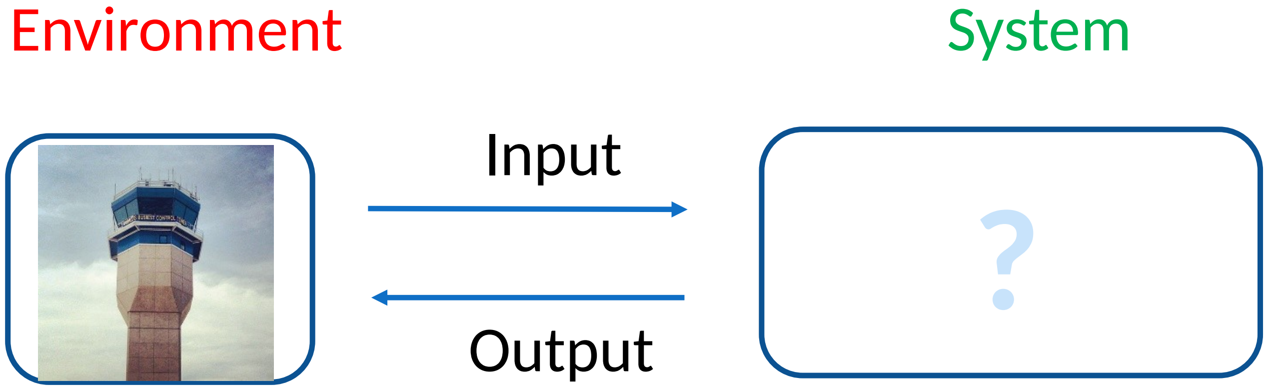
Reactive Synthesis



Given: linear temporal logic specification φ over inputs and outputs vars

Objective: synthesize a (reactive) system that will meet the specification

Reactive Synthesis - Example



At every time step: if *control* requests data then either grant now or grant at the next time

Reactive Synthesis - Example

At every time step: if *control* requests data then either grant now or grant at the next time

Temporal logic:

$$\textbf{\textit{always}}\ (r \rightarrow (g \vee \textbf{\textit{next}}\ (g)))$$

r, g – propositional variables

Reactive Synthesis - Example

At every time step: if *control* requests data then either grant now or grant at the next time

Temporal logic:

$$\textbf{\textit{always}}\ (r \rightarrow (g \vee \textbf{\textit{next}}\ (g)))$$

	i = 0	i = 1	i = 2	i = 3	...
env	r	r	¬ r	¬ r	...
sys	g	¬ g	g	¬ g	...

r, g – propositional variables

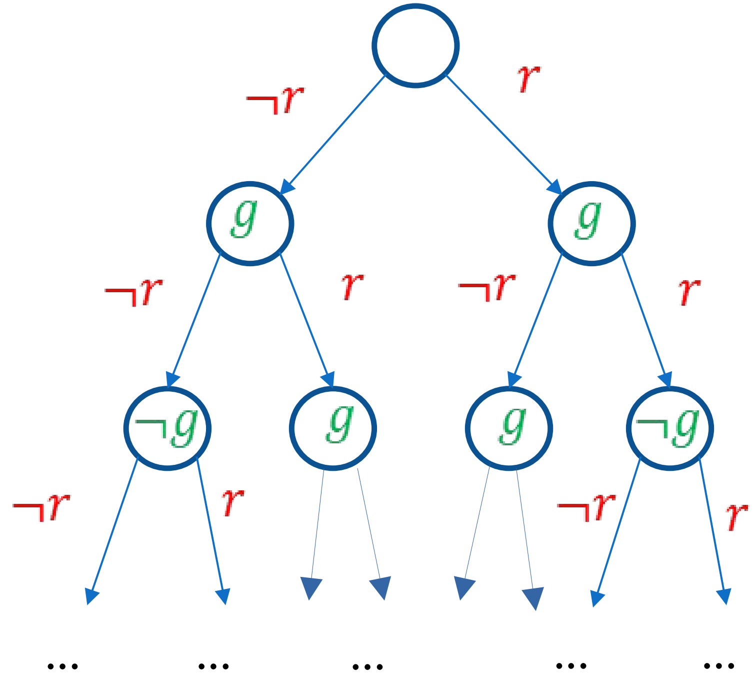
Reactive Synthesis - Example

At every time step: if *control* requests data then either grant now or grant at the next time

Temporal logic:

$$\textbf{\textit{always}}\ (r \rightarrow (g \lor \textbf{\textit{next}}\ (g)))$$

r, g – propositional variables

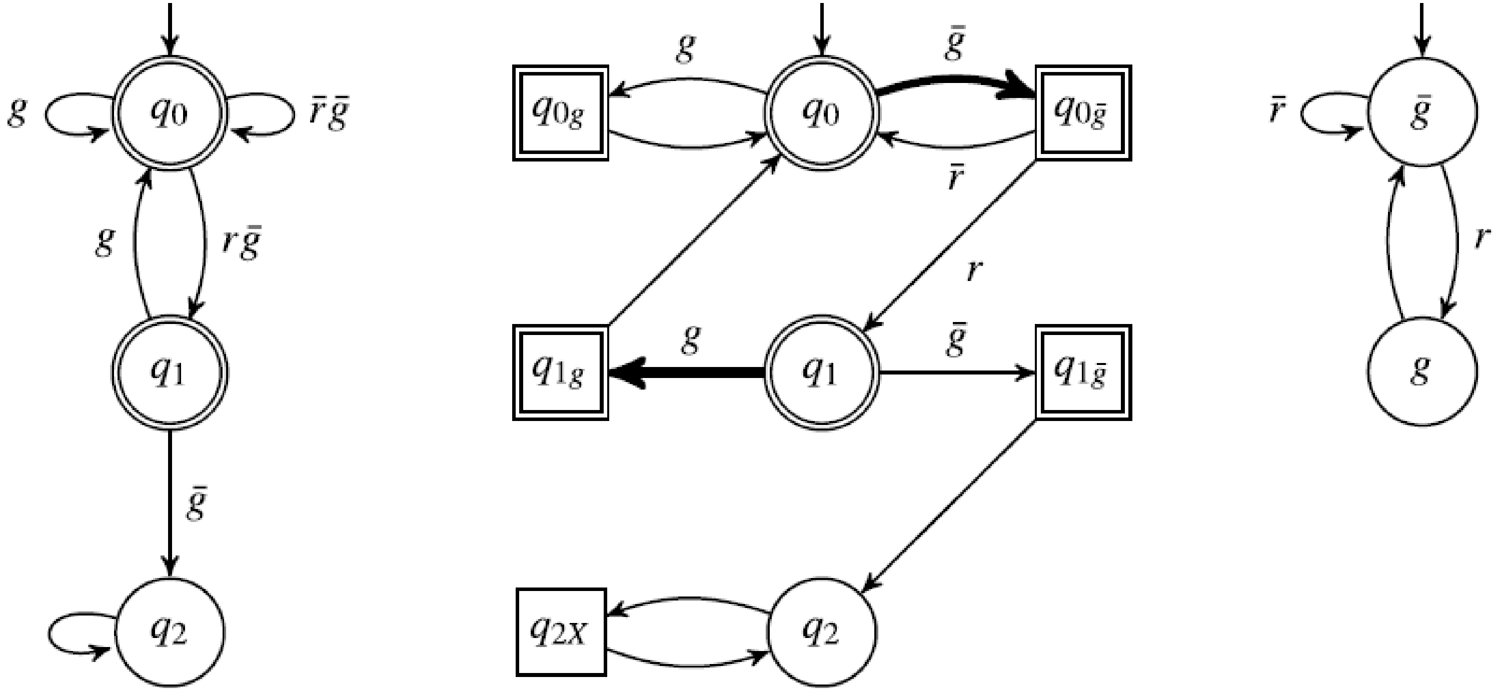


Strategy Tree

Reactive Synthesis Flow

Specification (LTL) → **Non-det (Büchi) Automaton** *Exponential Blowup* → **Deterministic game graph** *Exponential Blowup* → **Extract winning strategy** *(Poly- or quasi-poly-time)*

always $(r \rightarrow (g \vee$ ***next*** $(g)))$



(a) (b) (c)

Mealy Machine

Dependency in LTL

$$\varphi = \neg x \wedge Always(Next\ x \leftrightarrow (y_1 \wedge Next\ y_2)) \wedge \psi(x, z, y_1, y_2, y_3)$$

Dependency in LTL

$$\varphi = \neg x \wedge Always(Next\ x \leftrightarrow (y_1 \wedge Next\ y_2)) \wedge \psi\ (x, z, y_1, y_2, y_3)$$

$i = 0$ $x[i] = 0$

$i > 0$ $x[i]$ always assigned to $y_2[i] \wedge y_1[i - 1]$

Why bother?

- 300 out of 1141 SYNTCOMP (2023) benchmarks have at least 1 dependent output
  - Average of 41% dependent outputs in the 300 benchmarks
  - 26 benchmarks where all outputs are dependent

Why bother?

- 300 out of 1141 SYNTCOMP (2023) benchmarks have at least 1 dependent output
  - Average of 41% dependent outputs in the 300 benchmarks
  - 26 benchmarks where all outputs are dependent

Example from SYNTCOMP - **ltl2dpa14**

$$FG\left(\neg a \rightarrow (GF\mathbf{p_0} \vee (GFp_2 \wedge \neg GFp_1))\right) \wedge G((\mathbf{p_0} \wedge \neg p_1 \wedge \neg p_2) \vee (\neg \mathbf{p_0} \wedge p_1 \wedge \neg p_2) \vee (\neg \mathbf{p_0} \wedge \neg p_1 \wedge p_2))$$

$\mathbf{p_0}$ is dependent on $\{p_1, p_2\}$

Why bother?

- 300 out of 1141 SYNTCOMP (2023) benchmarks have at least 1 dependent output
  - Average of 41% dependent outputs in the 300 benchmarks
  - 26 benchmarks where all outputs are dependent

Example from SYNTCOMP - **ltl2dpa14**

$$FG\left(\neg a \rightarrow \left(GF\mathbf{p_0} \vee (GFp_2 \wedge \neg GFp_1)\right)\right) \wedge G\left((\mathbf{p_0} \wedge \neg p_1 \wedge \neg p_2) \vee (\neg \mathbf{p_0} \wedge p_1 \wedge \neg p_2) \vee (\neg \mathbf{p_0} \wedge \neg p_1 \wedge p_2)\right)$$

$\mathbf{p_0}$ is dependent on $\{p_1, p_2\}$

- Dependency in **Boolean Functional Synthesis**
  - [Akshay et al '18, '19, '20, '23; Golia et al'20, '21, '23; Mengel and Slivovsky'21; Peitl et al'19]
  - Tools: Manthan, BFSS

Why bother?

- 300 out of 1141 SYNTCOMP (2023) benchmarks have at least 1 dependent output
  - Average of 41% dependent outputs in the 300 benchmarks
  - 26 benchmarks where all outputs are dependent

Example from SYNTCOMP - **ltl2dpa14**

$$FG\left(\neg a \rightarrow (GF\mathbf{p_0} \vee (GFp_2 \wedge \neg GFp_1))\right) \wedge G((\mathbf{p_0} \wedge \neg p_1 \wedge \neg p_2) \vee (\neg \mathbf{p_0} \wedge p_1 \wedge \neg p_2) \vee (\neg \mathbf{p_0} \wedge \neg p_1 \wedge p_2))$$

$\mathbf{p_0}$ is dependent on $\{p_1, p_2\}$

- Dependency significantly helps **Boolean Functional Synthesis**
  - [Akshay et al '18, '19, '20, '23; Golia et al'20, '21, '23; Mengel and Slivovsky'21; Peitl et al'19]
  - Tools: Manthan, BFSS
- Can we lift ideas from **Boolean Functional Synthesis** to **Reactive Synthesis** ?
  - Boolean Synthesis by I/O Separation [CFTY'21] → LTL/LTLf synthesis [ABFTYW'21, DFPZ'23]

Research Questions:

- How do we formally define dependency in reactive synthesis?
- How do we find dependent variables?
- How do we exploit dependency in reactive synthesis?
- Do experiments confirm the dependency benefits?

Dependency in Boolean Formulas

In a Boolean formula $F(x, y_1, \ldots y_k)$, x is dependent on $Y \subseteq \{y_1, \ldots y_k\}$ if

For every two satisfying assignments σ, σ' for F

if $\sigma|_Y = \sigma'|_Y$ then $\sigma|_x = \sigma'|_x$

Dependency in Boolean Formulas

$$F = (x \leftrightarrow (y_1 \wedge y_2)) \wedge (x \vee y_3)$$

Then x is dependent on $\{y_1, y_2\}$ in F

Finding dependent variables in Boolean formulas is not always obvious.

Dependency in Boolean Formulas

$$F = (x \leftrightarrow (y_1 \wedge y_2)) \wedge (x \vee y_3)$$

Then x is dependent on $\{y_1, y_2\}$ in F

Finding dependent variables in Boolean formulas is not always obvious.

Dependencies can exist even without syntactic equivalences

$(y_1 \vee \neg x_2) \wedge (x_2 \vee \neg y_1) \wedge ((x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg y_1))$: x_2 dependent on x_1

Dependency in LTL

In an LTL formula, $\varphi(x, y_1 \ldots y_k)$, x is <u>dependent</u> on $Y \subseteq \{y_1 \ldots y_k\}$ if:

Dependency in LTL

In an LTL formula, $\varphi(x, y_1 \dots y_k)$, x is <u>dependent</u> on $Y \subseteq \{y_1 \dots y_k\}$ if:

For every two infinite words w, w' that satisfy φ,

For every $i \geq 0$

If $w[0 \dots i-1] = w'[0 \dots i-1]$ and $w[i]|_Y = w'[i]|_Y$

then $w[i]|_x = w'[i]|_x$

($w[0,-1]$ is the empty word)

Non-Dependent Variables

In an LTL formula, $\varphi(x, y_1 \dots y_k)$, x is non-dependent on $Y \subseteq \{y_1 \dots y_k\}$ if:

There exist two infinite words w, w' that satisfy φ,

there exists $i \geq 0$ s.t.

$w[0 \dots i-1] = w'[0 \dots i-1]$,$w[i]|_Y = w'[i]|_Y$

and $w[i]|_x \neq w'[i]|_x$

Dependency in LTL - Example

$$\varphi = \neg x \wedge Always\big(Next\ x \leftrightarrow (y_1 \wedge Next\ y_2)\big) \wedge Finally\ (y_3)$$

x is dependent on $\{y_1, y_2\}$

Dependency in LTL - Example

$$\varphi = \neg x \wedge Always(Next\ x \leftrightarrow (y_1 \wedge Next\ y_2)) \wedge Finally\ (y_3)$$

x is dependent on $\{y_1, y_2\}$

Accepting trace of φ:

	i=0	i=1	i=2	i=3	i=4	…
x	0					…
y_1	0					…
y_2	0					…
y_3	0					…

Dependency in LTL - Example

$$\varphi = \neg x \wedge Always(Next\ x \leftrightarrow (y_1 \wedge Next\ y_2)) \wedge Finally\ (y_3)$$

x is dependent on $\{y_1, y_2\}$

Accepting trace of φ:

	i=0	i=1	i=2	i=3	i=4	…
x	0	0				…
y_1	0	1				…
y_2	0	1				…
y_3	0	0				…

Dependency in LTL - Example

$$\varphi = \neg x \wedge Always(Next\ x \leftrightarrow (y_1 \wedge Next\ y_2)) \wedge Finally\ (y_3)$$

x is dependent on $\{y_1, y_2\}$

Accepting trace of φ:

	i=0	i=1	i=2	i=3	i=4	…
x	0	0	1			…
y_1	0	1	1			…
y_2	0	1	1			…
y_3	0	0	0			…

Dependency in LTL - Example

$$\varphi = \neg x \wedge Always(Next\ x \leftrightarrow (y_1 \wedge Next\ y_2)) \wedge Finally\ (y_3)$$

x is dependent on $\{y_1, y_2\}$

Accepting trace of φ:

	i=0	i=1	i=2	i=3	i=4	…
x	0	0	1	0		…
y_1	0	1	1	1		…
y_2	0	1	1	0		…
y_3	0	0	0	1		…

Dependency in LTL - Example

$$\varphi = \neg x \wedge Always(Next\ x \leftrightarrow (y_1 \wedge Next\ y_2)) \wedge Finally\ (y_3)$$

x is dependent on $\{y_1, y_2\}$

Accepting trace of φ:

	i=0	i=1	i=2	i=3	i=4	…
x	0	0	1	0	1	…
y_1	0	1	1	1	0	…
y_2	0	1	1	0	1	…
y_3	0	0	0	1	1	…

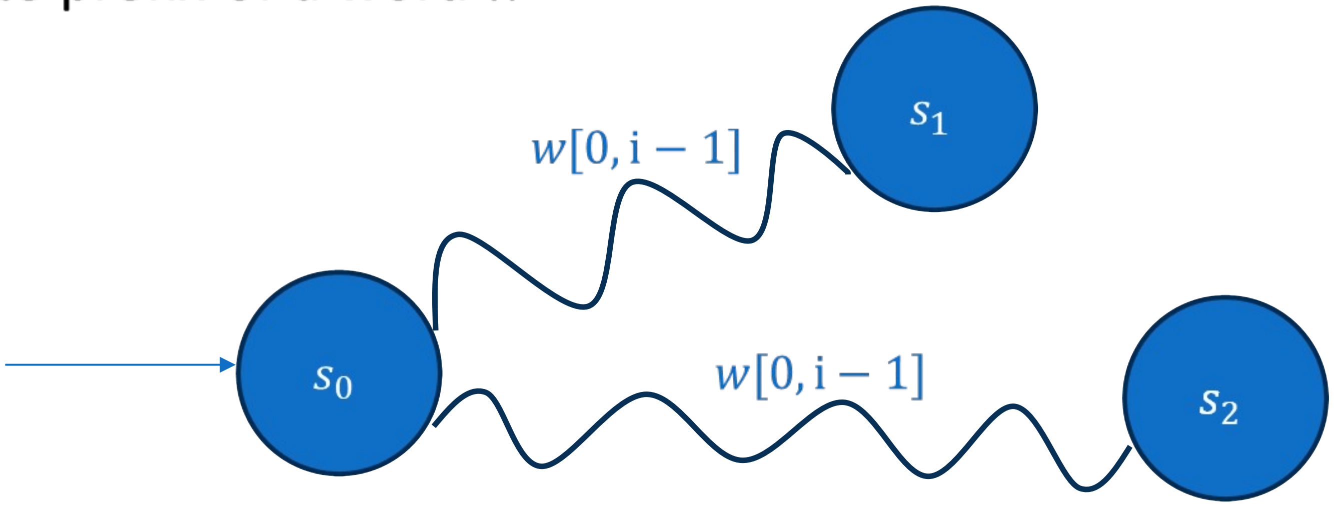
Finding Dependent Variables

LTL Specification φ → Non-deterministic Büchi Automaton A_{φ}

- Standard construction: $L(\varphi) = L(A_{\varphi})$
- Prune NBA A_{φ}: remove all states/edges that do not lead to accepting states.
- All our NBAs are pruned.

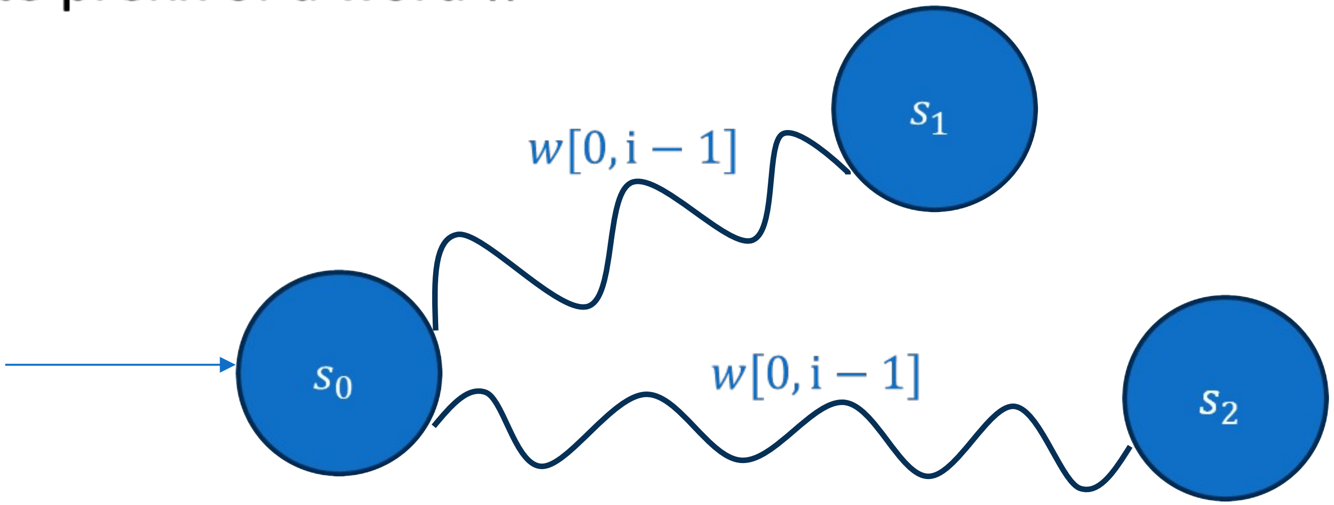
Finding Dependent Variables

States s_1, s_2 are compatible if both s_1 and s_2 can be reached from start state on same finite prefix of a word w



Finding Dependent Variables

States s_1, s_2 are compatible if both s_1 and s_2 can be reached from start state on same finite prefix of a word w

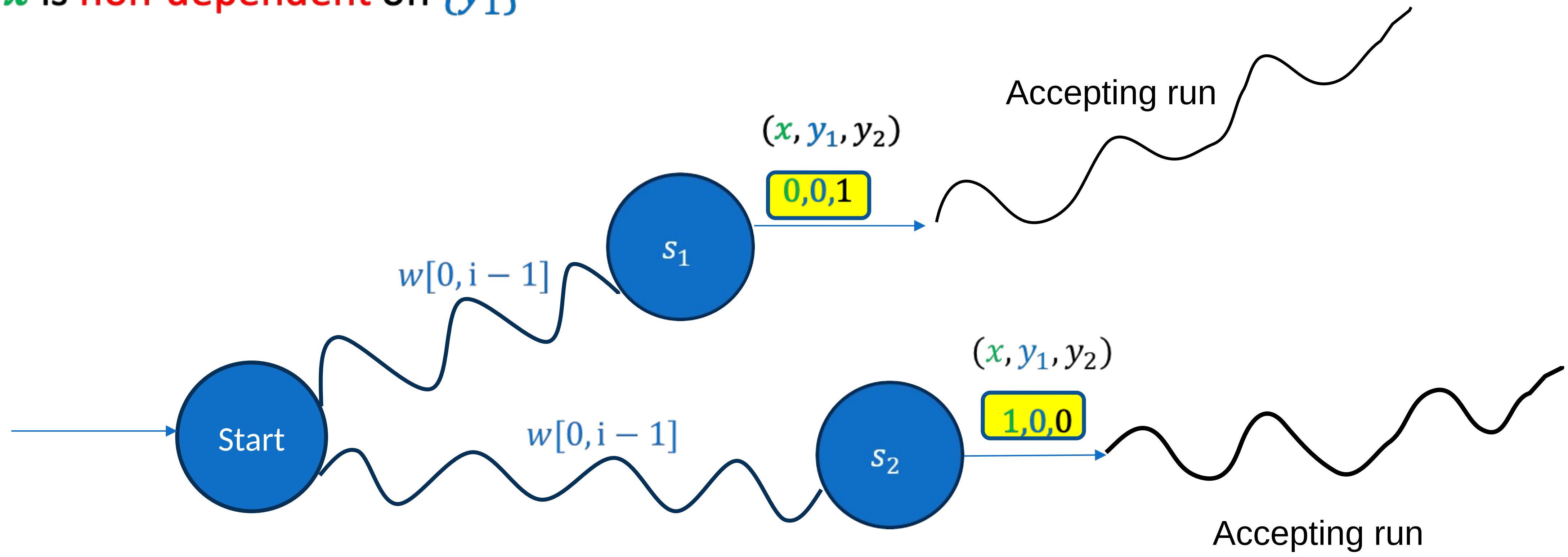


We find all (unordered) pairs of compatible states in A_φ

$\{(s_0, s_0), (s_1, s_1), (s_2, s_2), (s_1, s_2)\}$

Example of not-dependent variable

- x is non-dependent on $\{y_1\}$



Finding Dependent Variables

1. $Y = Vars(\varphi)$

2. Pick next variable $x \in Y$

3. For each pair of compatible states (s_1, s_2)

   If x is non-dependent on $Y \setminus \{x\}$ from (s_1, s_2) then go to step 2

4. Mark x as dependent on Y

5. $Y = Y \setminus \{x\}$, go to step 2

Finding Dependent Variables

1. $Y = Vars(\varphi)$
2. Pick next variable $x \in Y$
3. For each pair of compatible states (s_1, s_2)

   If x is non-dependent on $Y \backslash \{x\}$ from (s_1, s_2) then go to step 2
4. Mark x as dependent on Y
5. $Y = Y \backslash \{x\}$, go to step 2

- Gives a subset-maximal set of dependent variables.
- Order of variables in Step 2 is important.

(See paper for details)

Utilizing dependency in synthesis

LTL specification $\varphi(I, O)$ where I are the inputs and O are the outputs

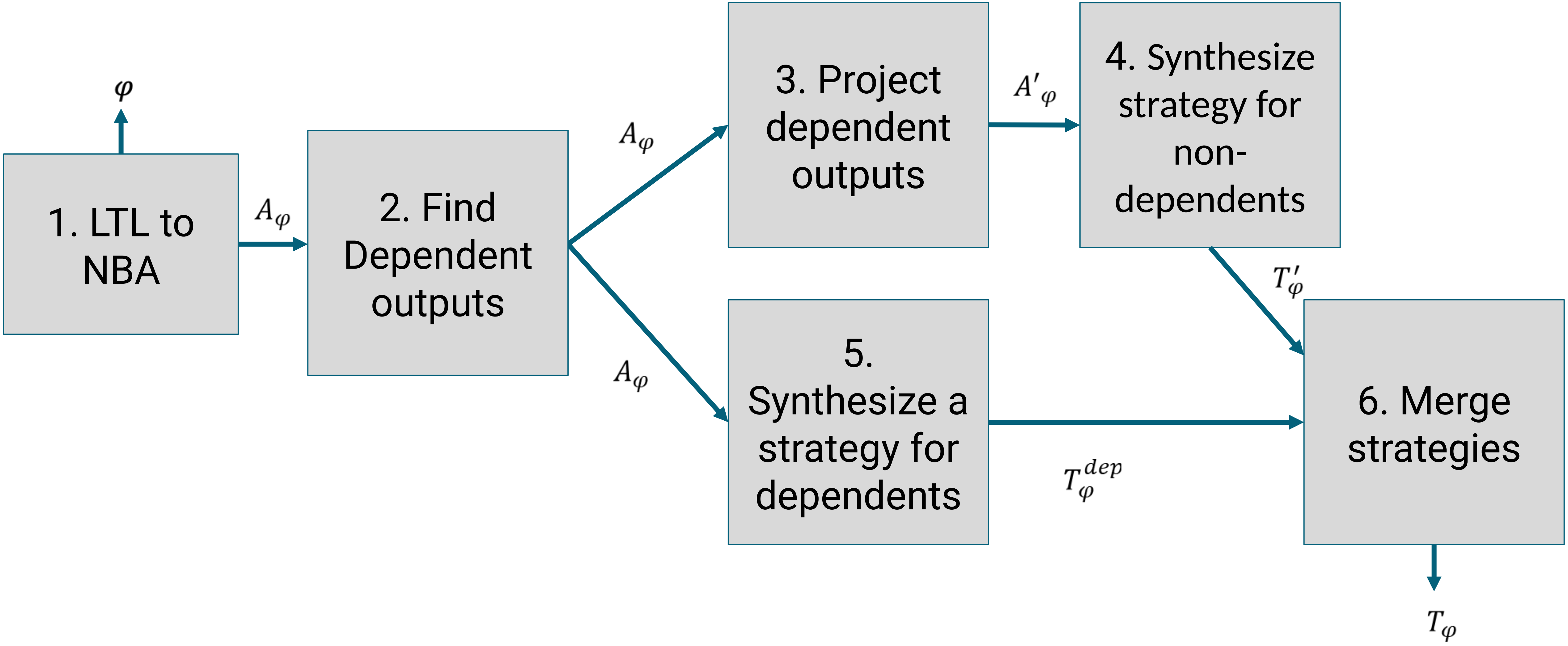
We focus on finding dependent output variables.

an output variable can be dependent on both input and other output variables

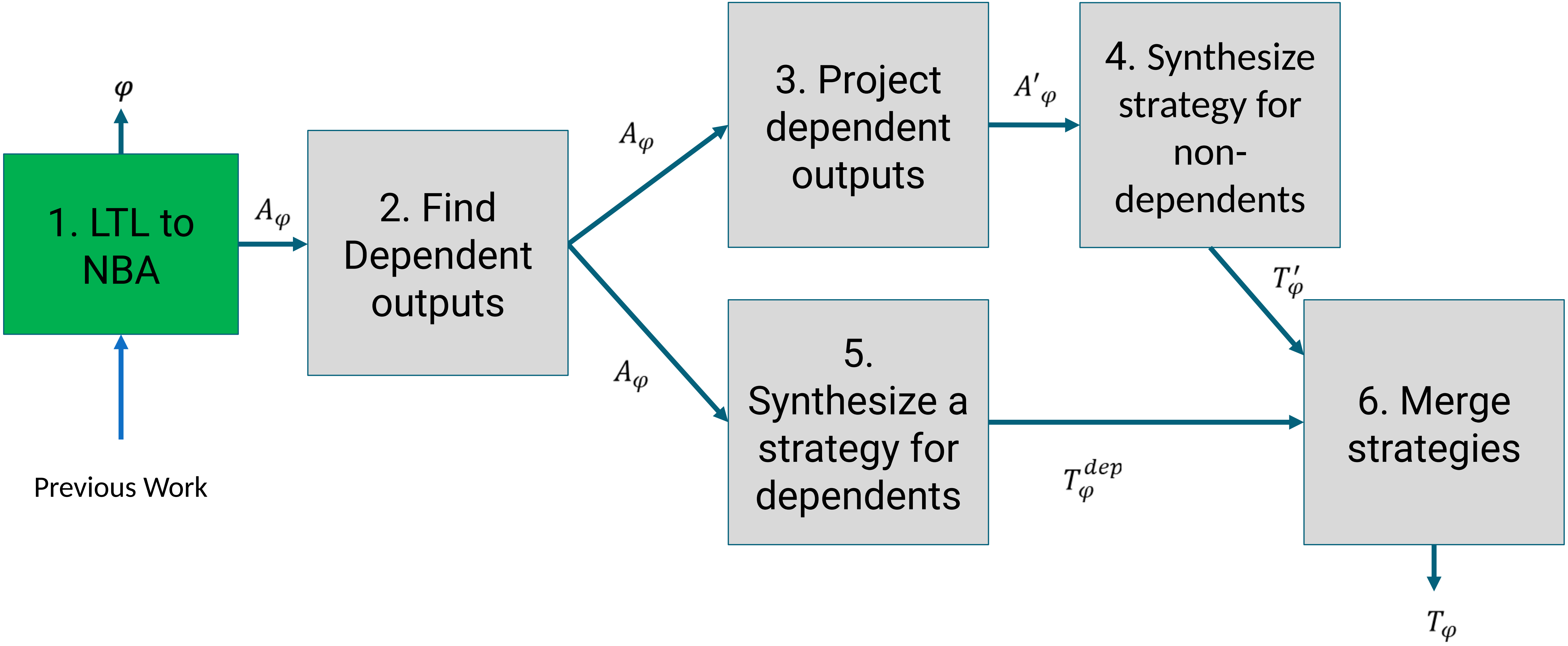
Synthesis Flow

LTL φ → NBA A_φ → (symbolic) Mealy Machine T_φ

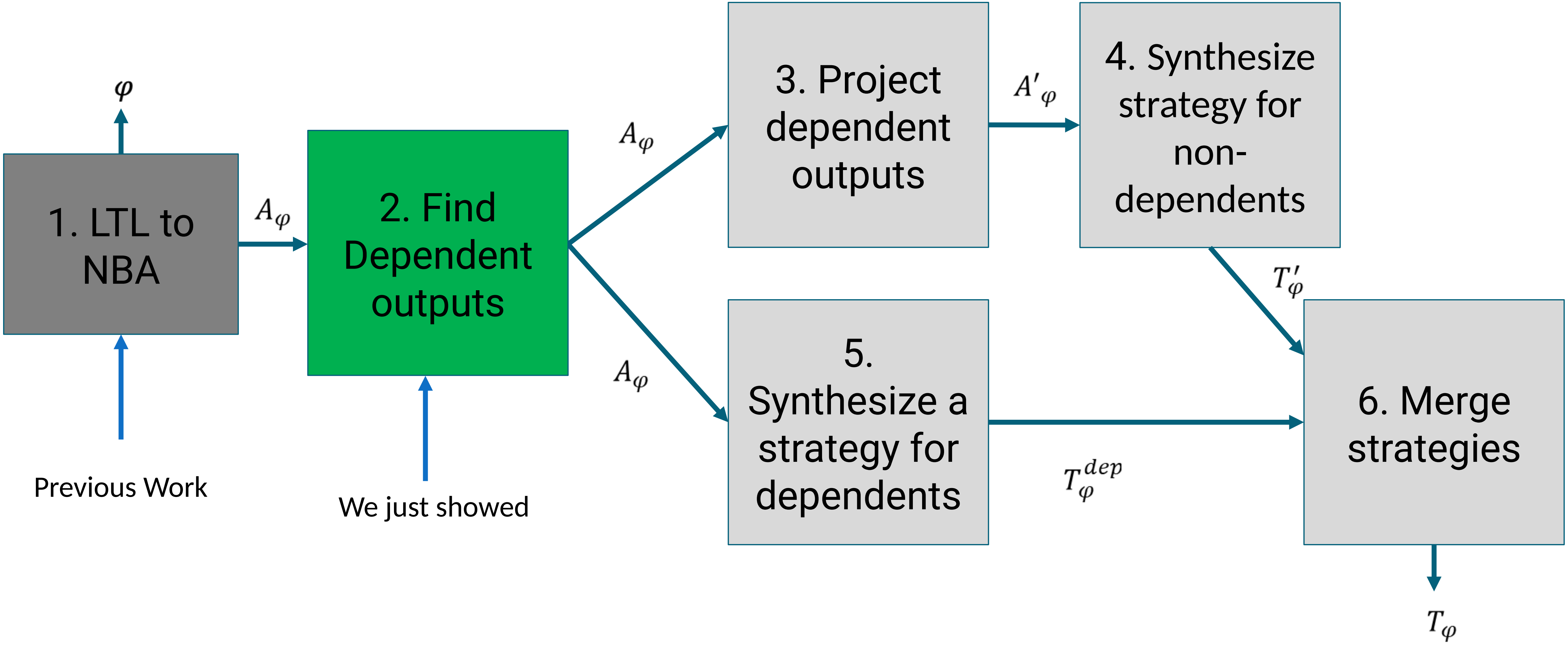
Synthesis Pipeline



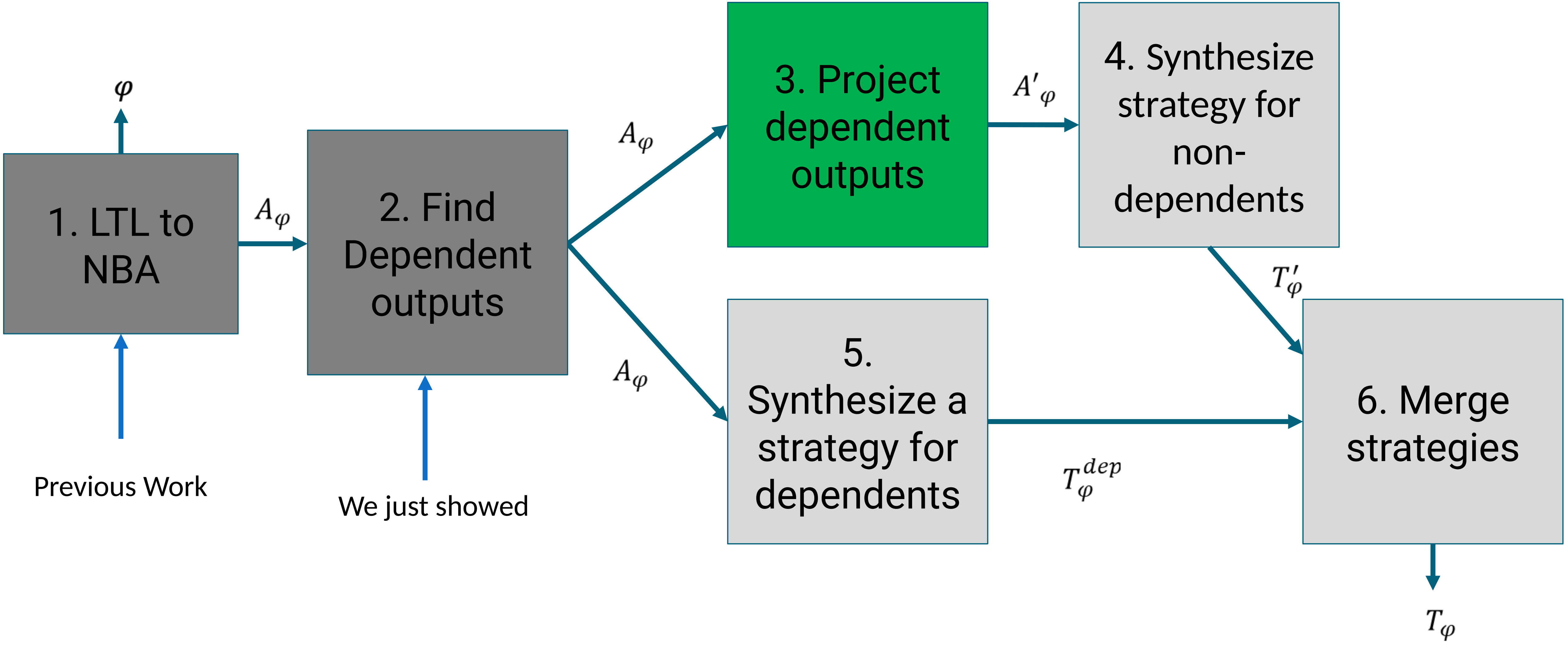
Synthesis Pipeline



Synthesis Pipeline



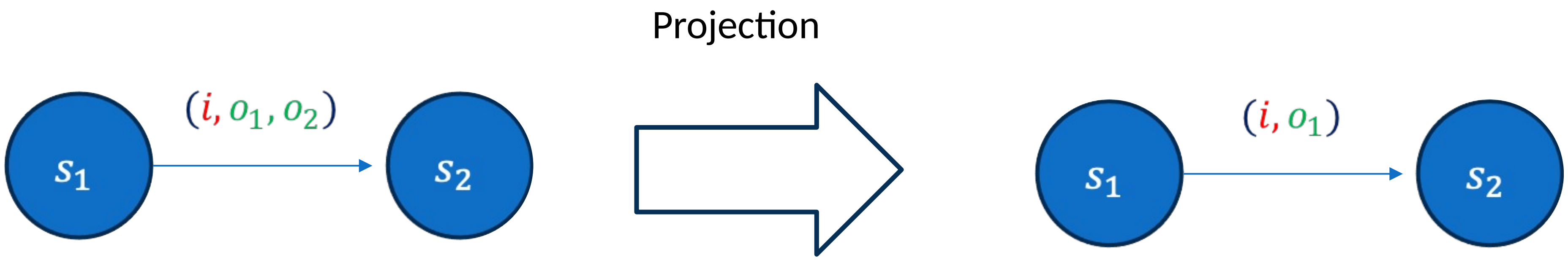
Synthesis Pipeline



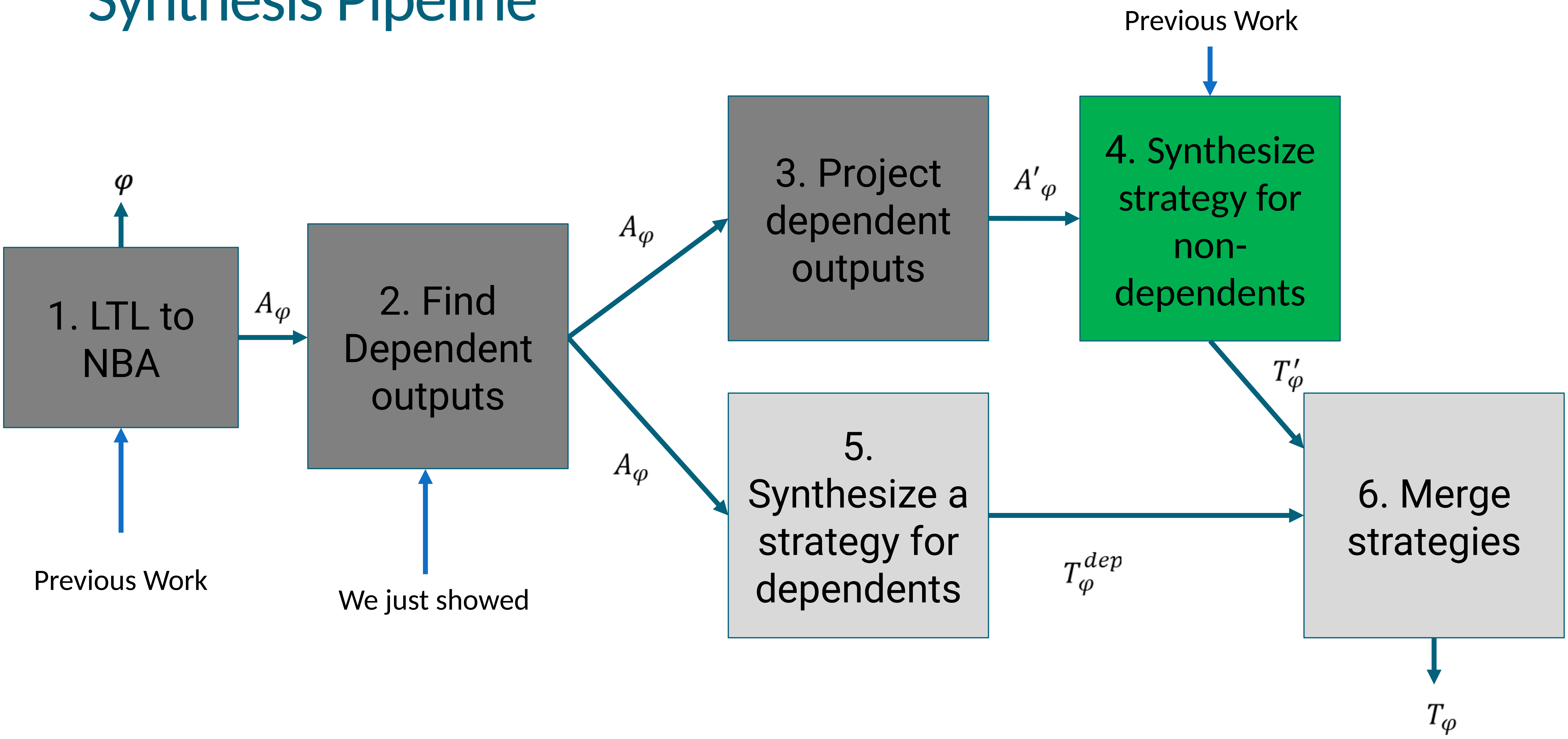
Step 3: Project dependent variables

The projection process removes the dependent variables from all NBA edges labels.

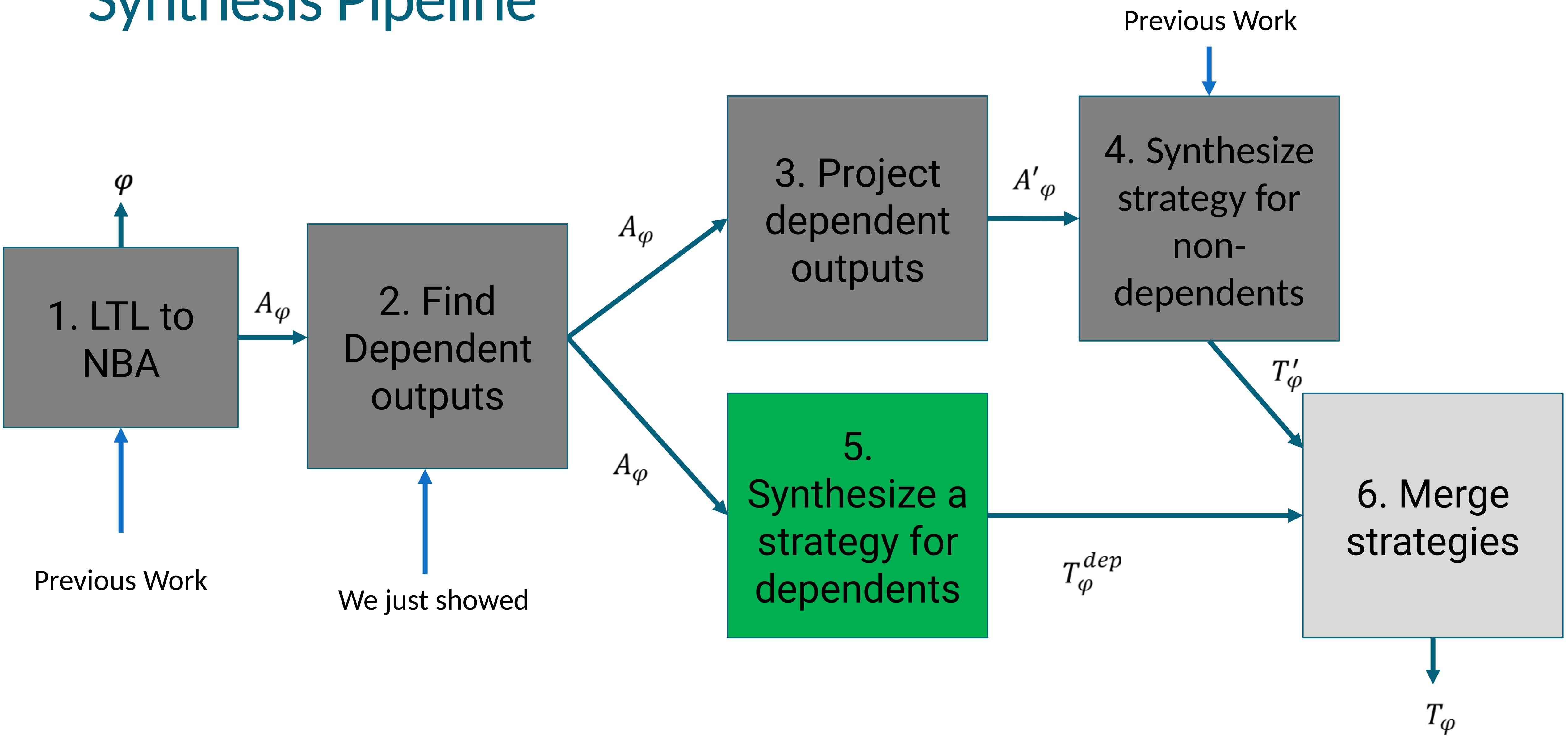
Assume o_2 is dependent on $\{i, o_1\}$ in $\varphi(i, o_1, o_2)$.



Synthesis Pipeline



Synthesis Pipeline



Step 5: Dependent variables synthesis

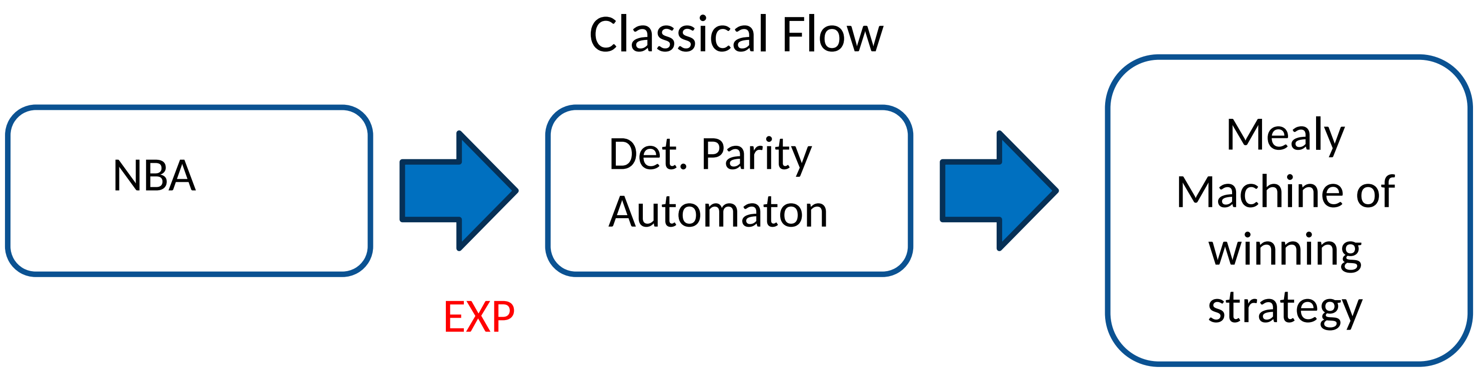
Inputs: original inputs and non-dependent output variables

Outputs: Symbolic Mealy machine for dependent output variables

Step 5: Dependent variables synthesis

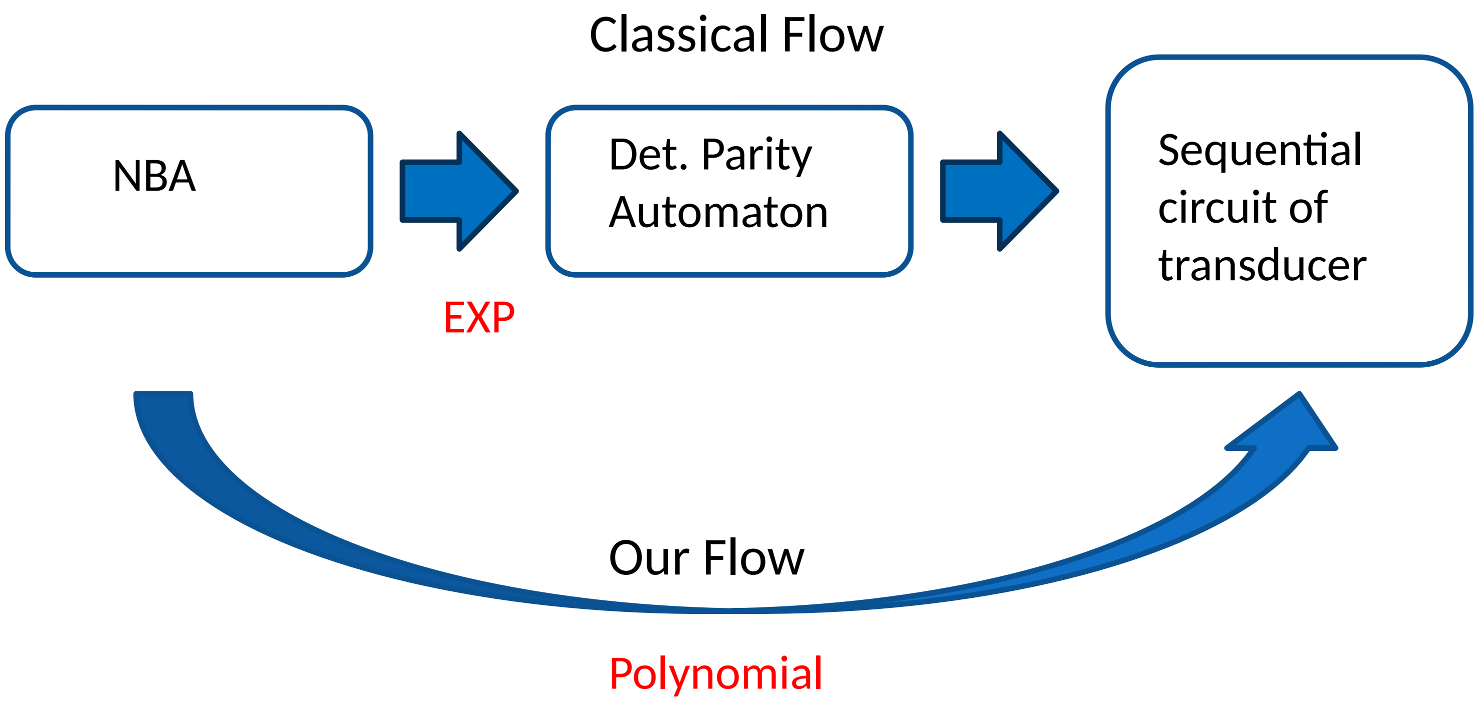
Inputs: original inputs and non-dependent output variables

Outputs: Symbolic Mealy machine for dependent output variables



Step 5: Dependent variables synthesis

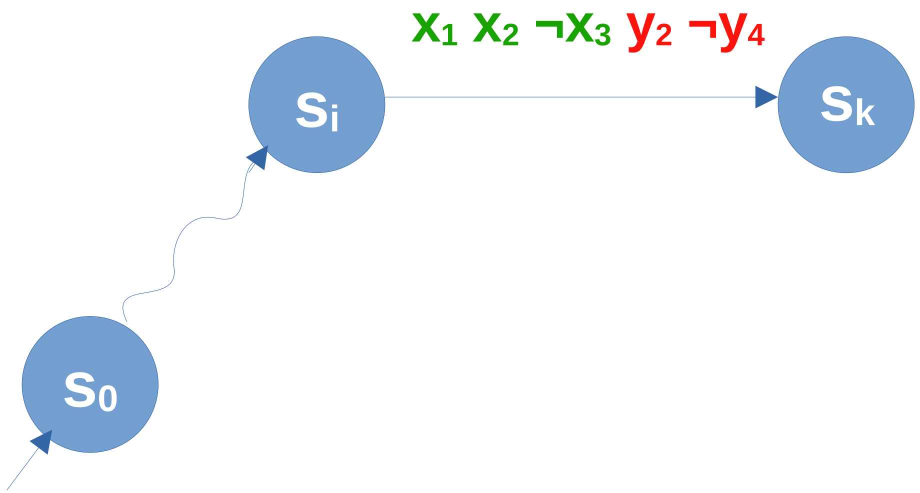
Inputs: original inputs and non-dependent output variables
Outputs: Symbolic Mealy machine for dependent output variables



Based on implicit subset-construction
(See details in the paper)

Step 5: Dependent variables synthesis

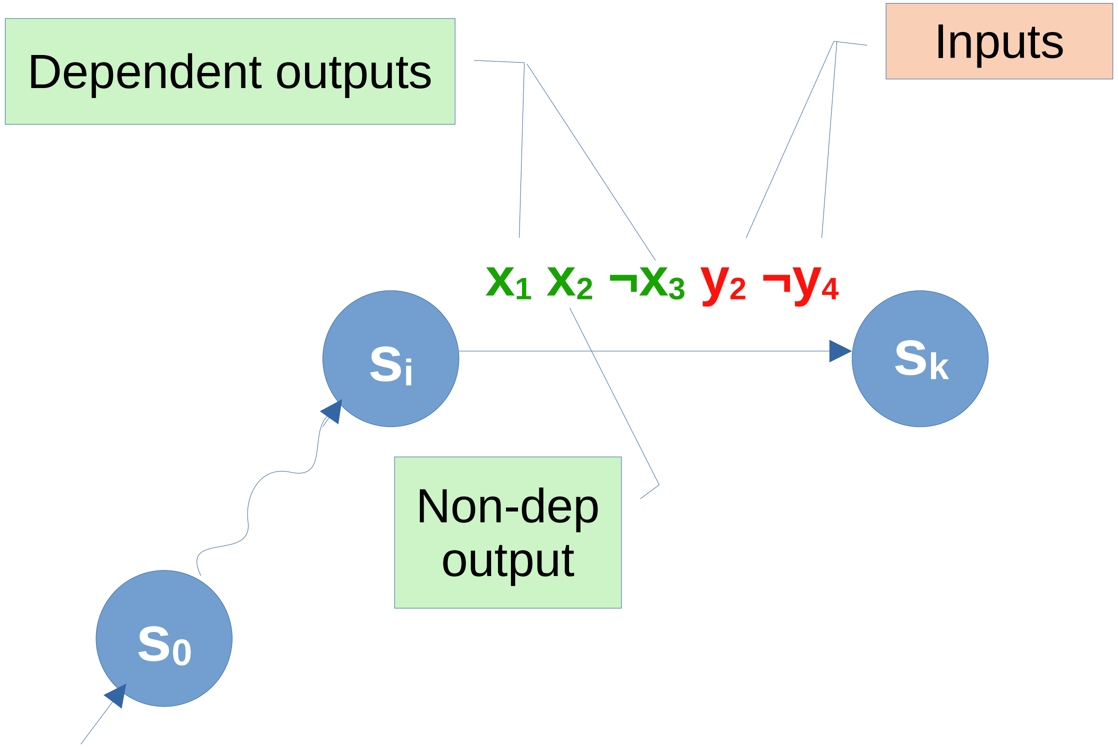
Inputs: original inputs and non-dependent output variables
Outputs: Symbolic Mealy machine for dependent output variables



Step 5: Dependent variables synthesis

Inputs: original inputs and non-dependent output variables

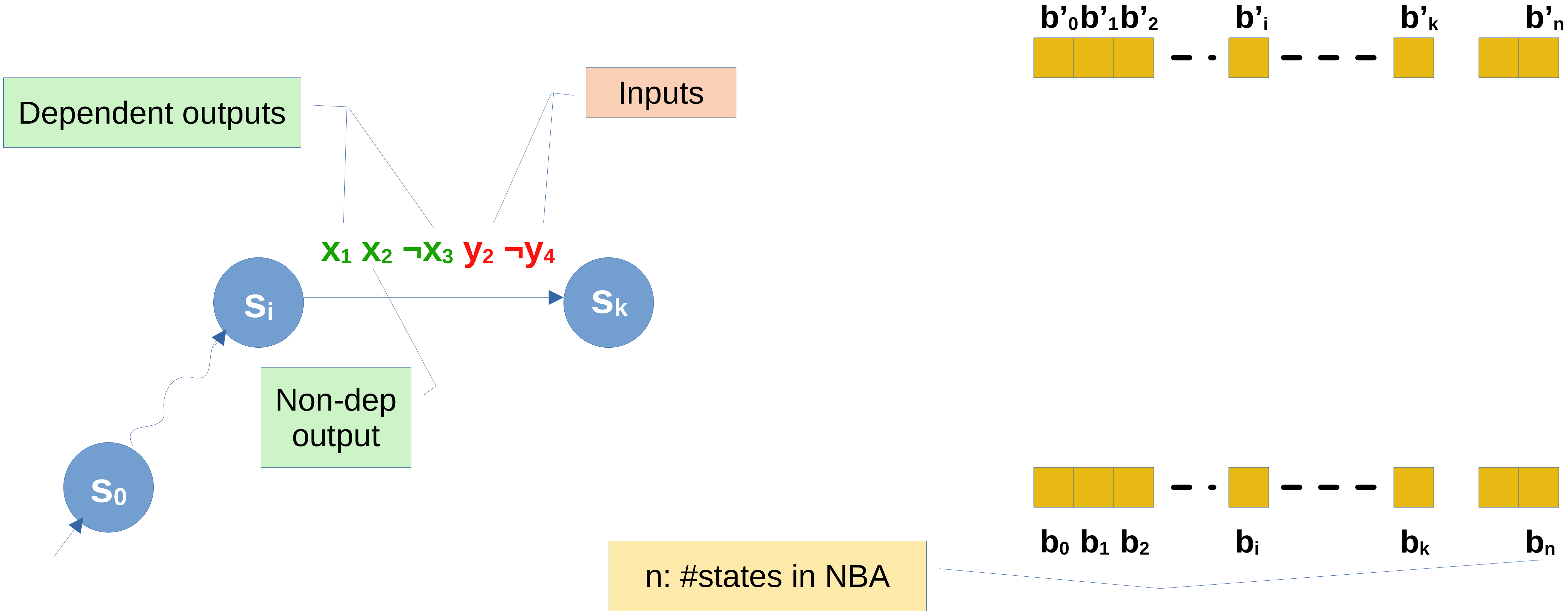
Outputs: Symbolic Mealy machine for dependent output variables



Step 5: Dependent variables synthesis

Inputs: original inputs and non-dependent output variables

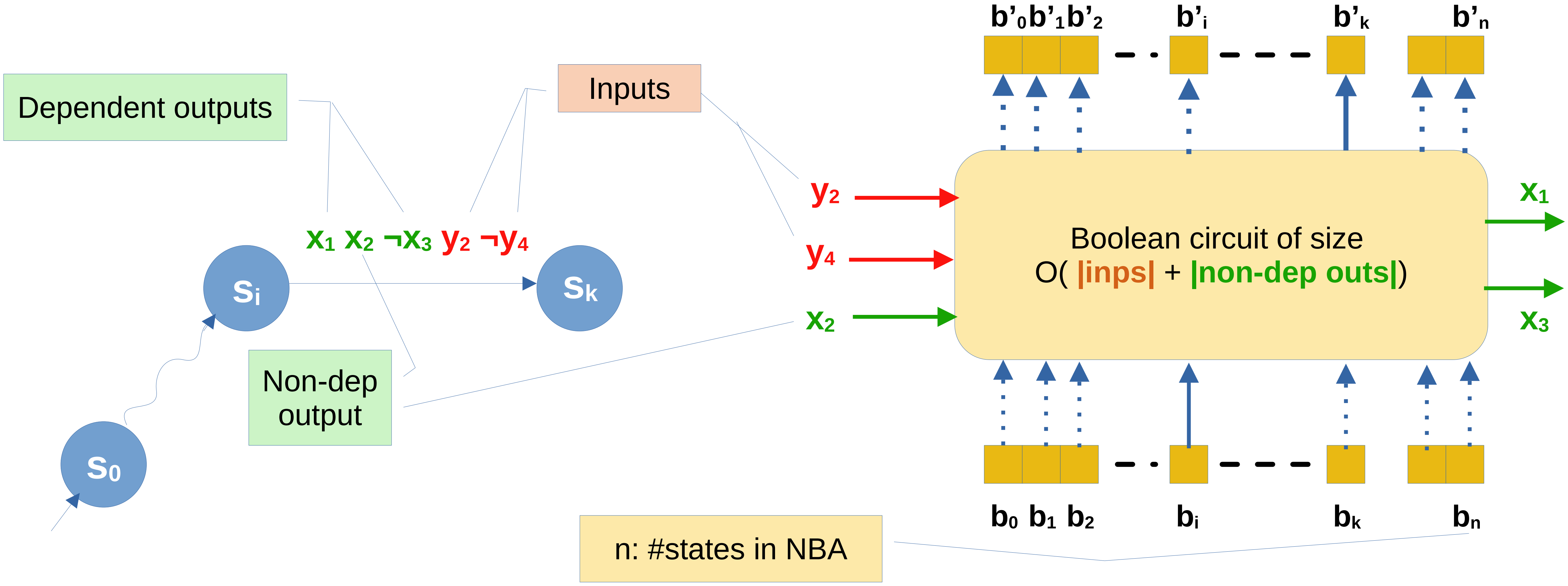
Outputs: Symbolic Mealy machine for dependent output variables



Step 5: Dependent variables synthesis

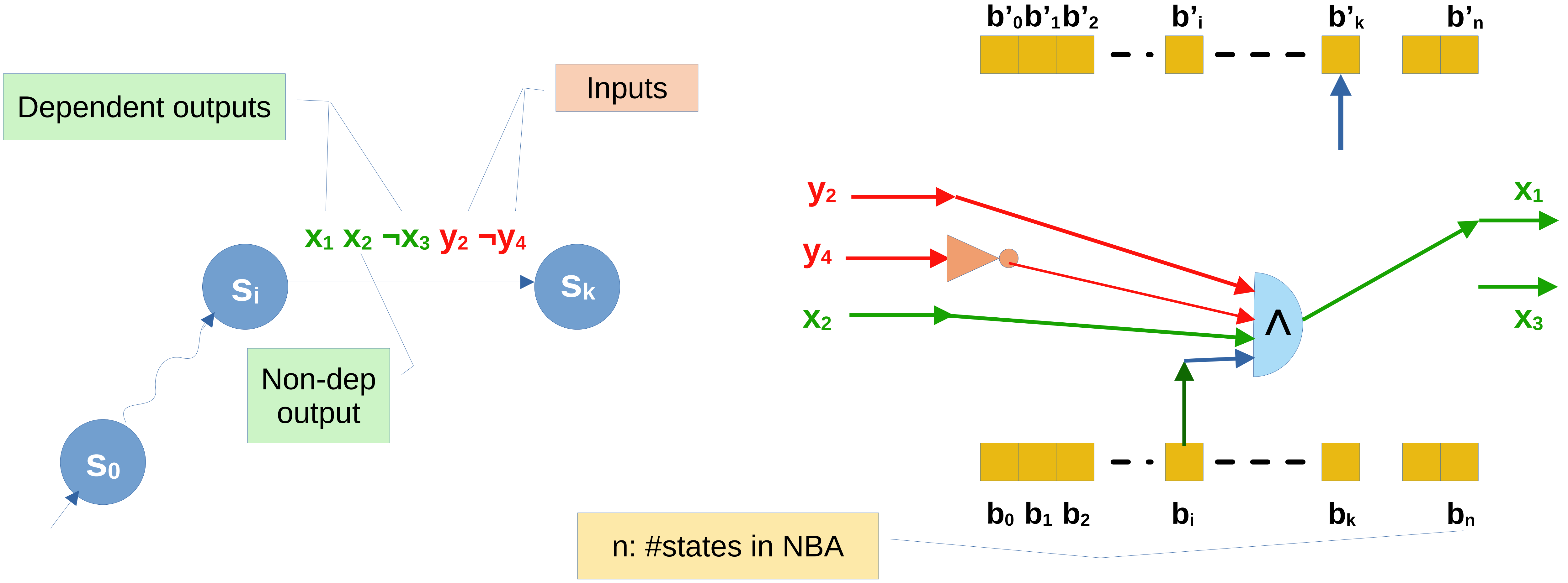
Inputs: original inputs and non-dependent output variables

Outputs: Symbolic Mealy machine for dependent output variables



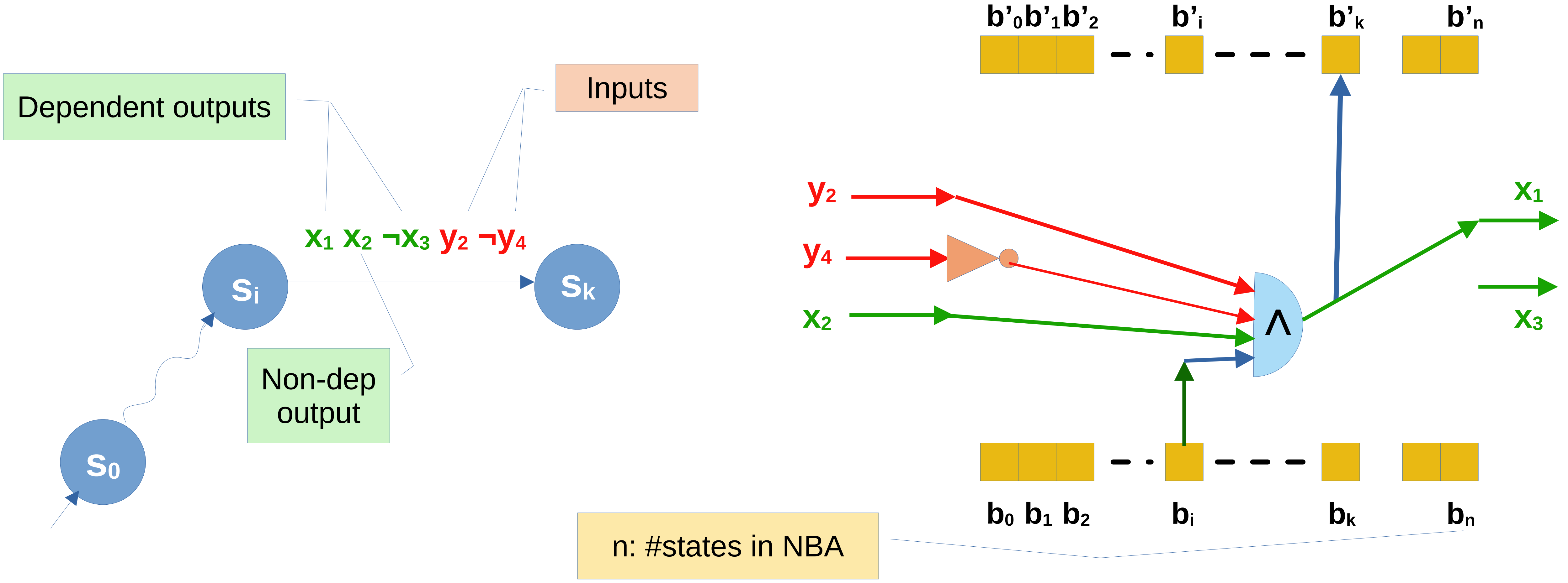
Step 5: Dependent variables synthesis

Inputs: original inputs and non-dependent output variables
Outputs: Symbolic Mealy machine for dependent output variables



Step 5: Dependent variables synthesis

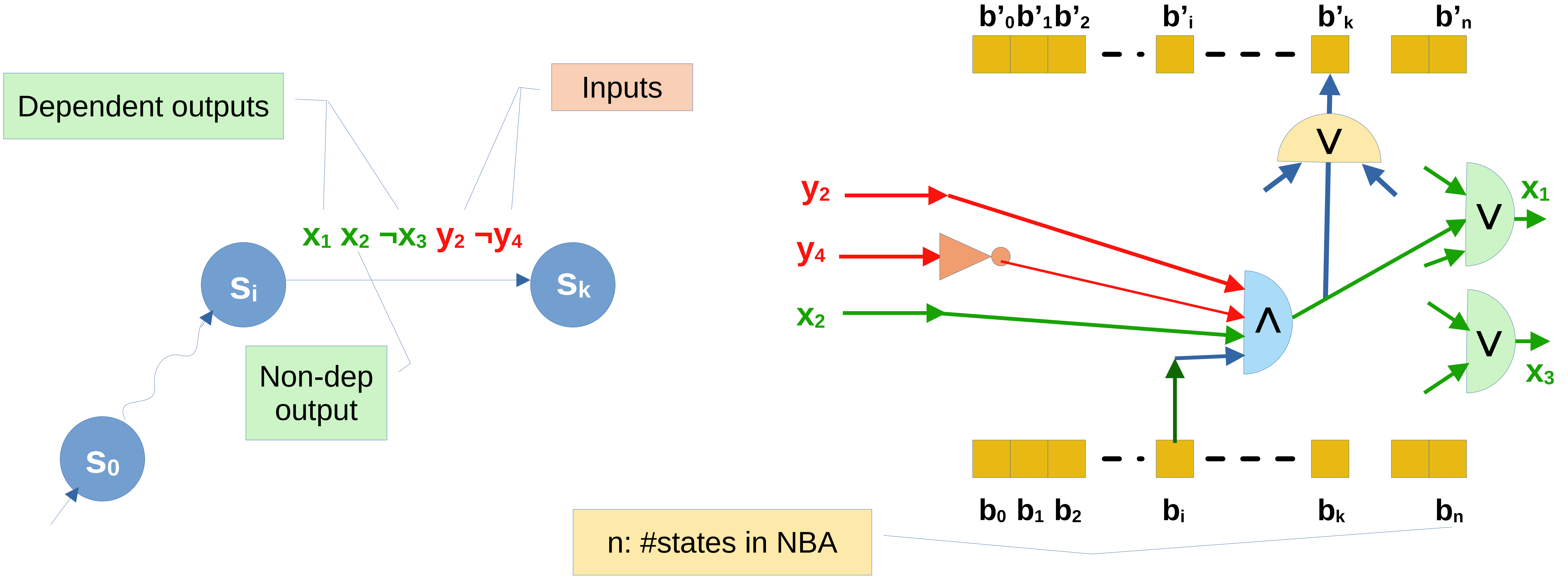
Inputs: original inputs and non-dependent output variables
Outputs: Symbolic Mealy machine for dependent output variables



Step 5: Dependent variables synthesis

Inputs: original inputs and non-dependent output variables

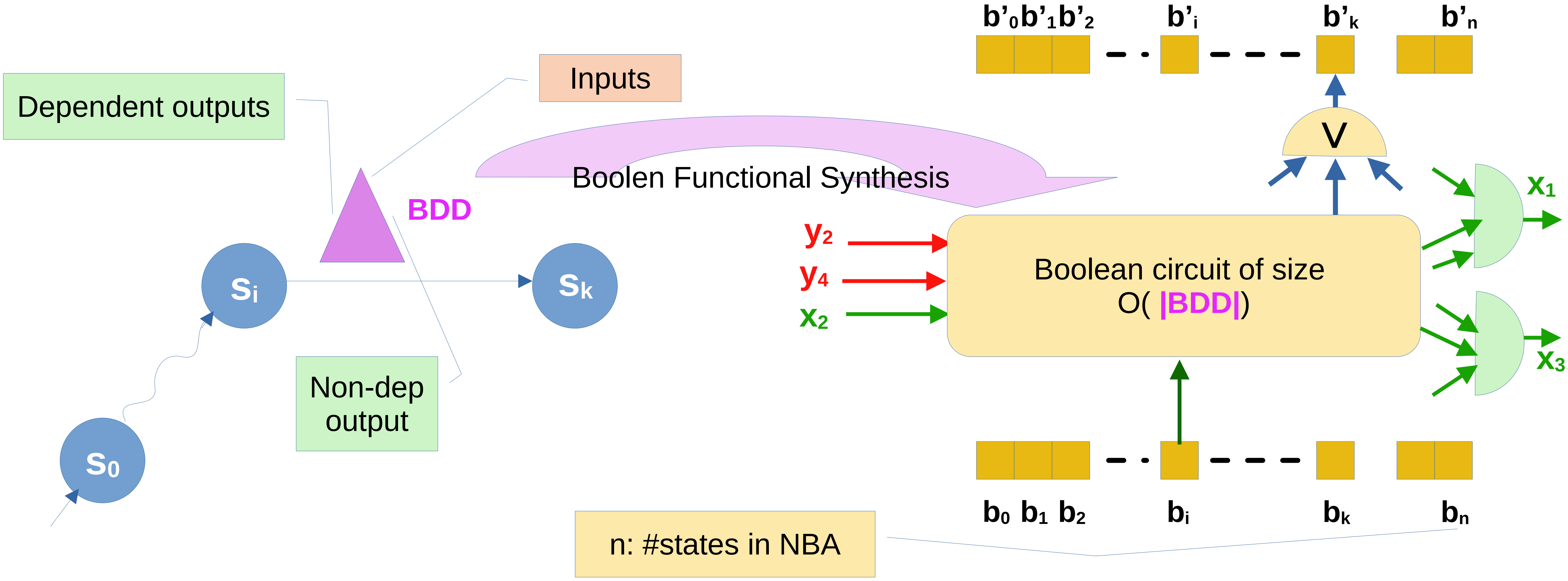
Outputs: Symbolic Mealy machine for dependent output variables



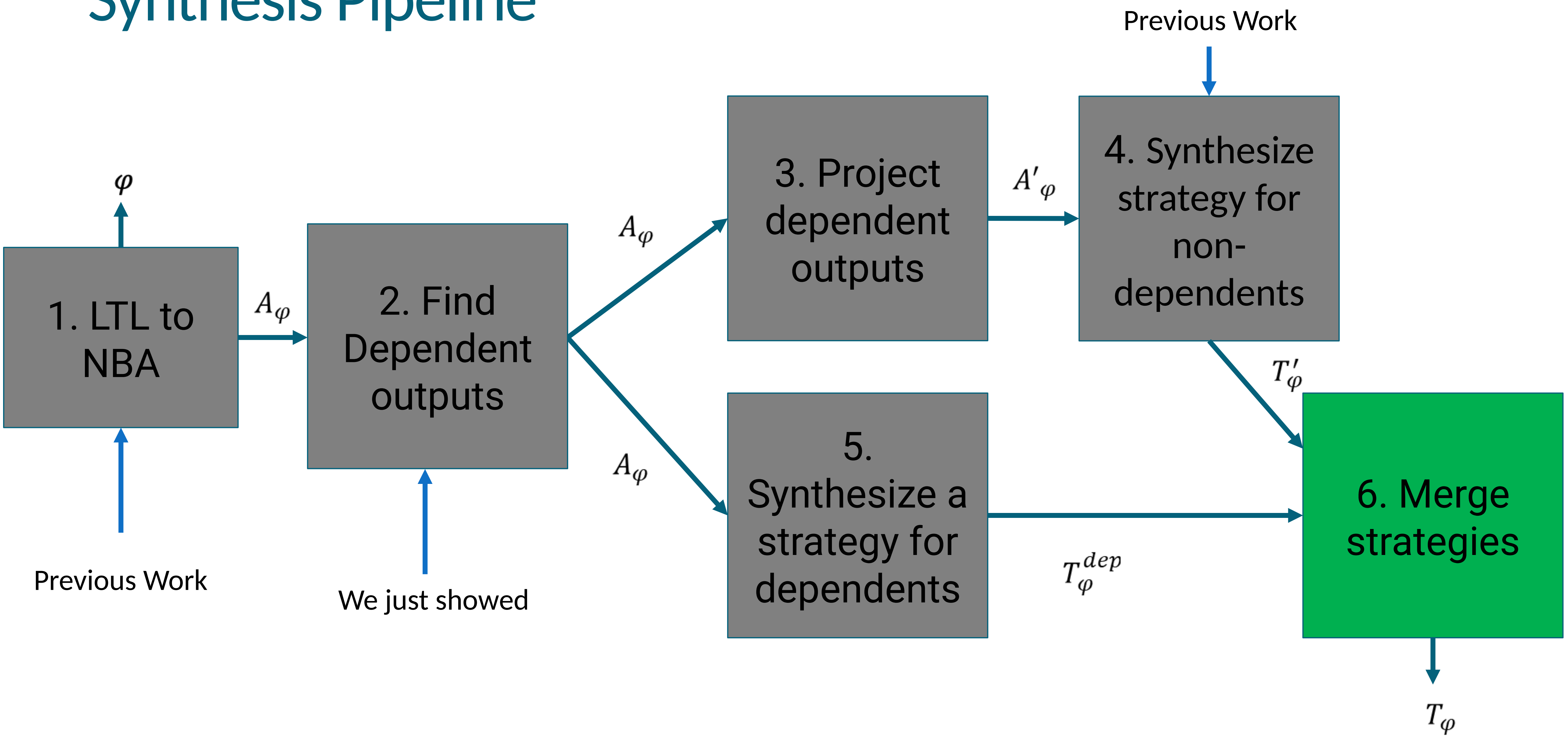
Step 5: Dependent variables synthesis

Inputs: original inputs and non-dependent output variables

Outputs: Symbolic Mealy machine for dependent output variables



Synthesis Pipeline

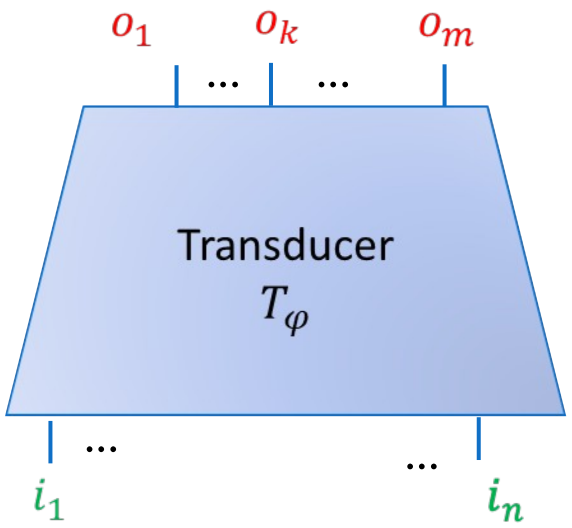


Step 6: Merge Transducers

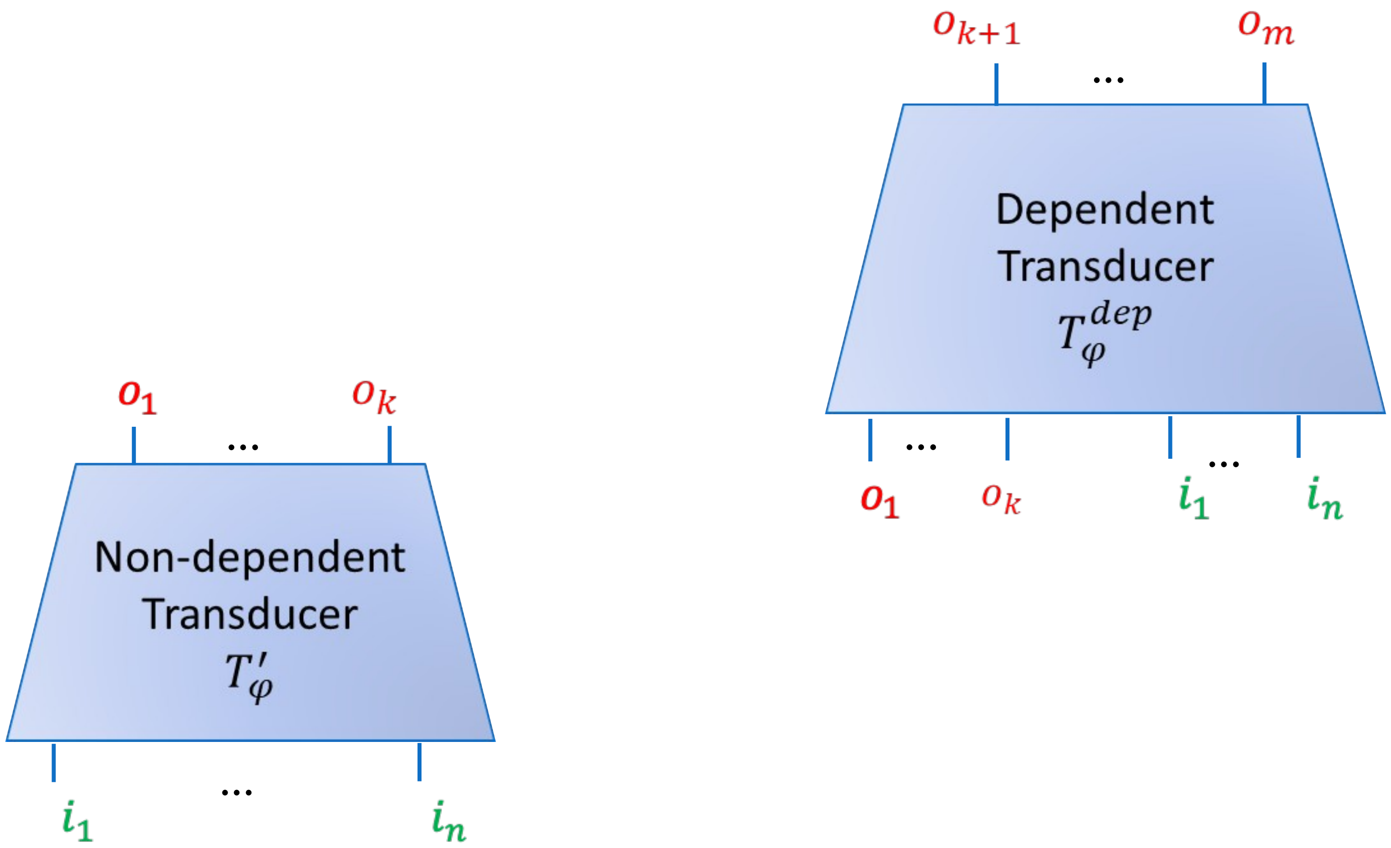
- Our transducers are described as a sequential circuits.

$o_1, \dots, o_k$ are non-dependent variables

$o_{k+1}, \dots, o_m$ are dependent variables

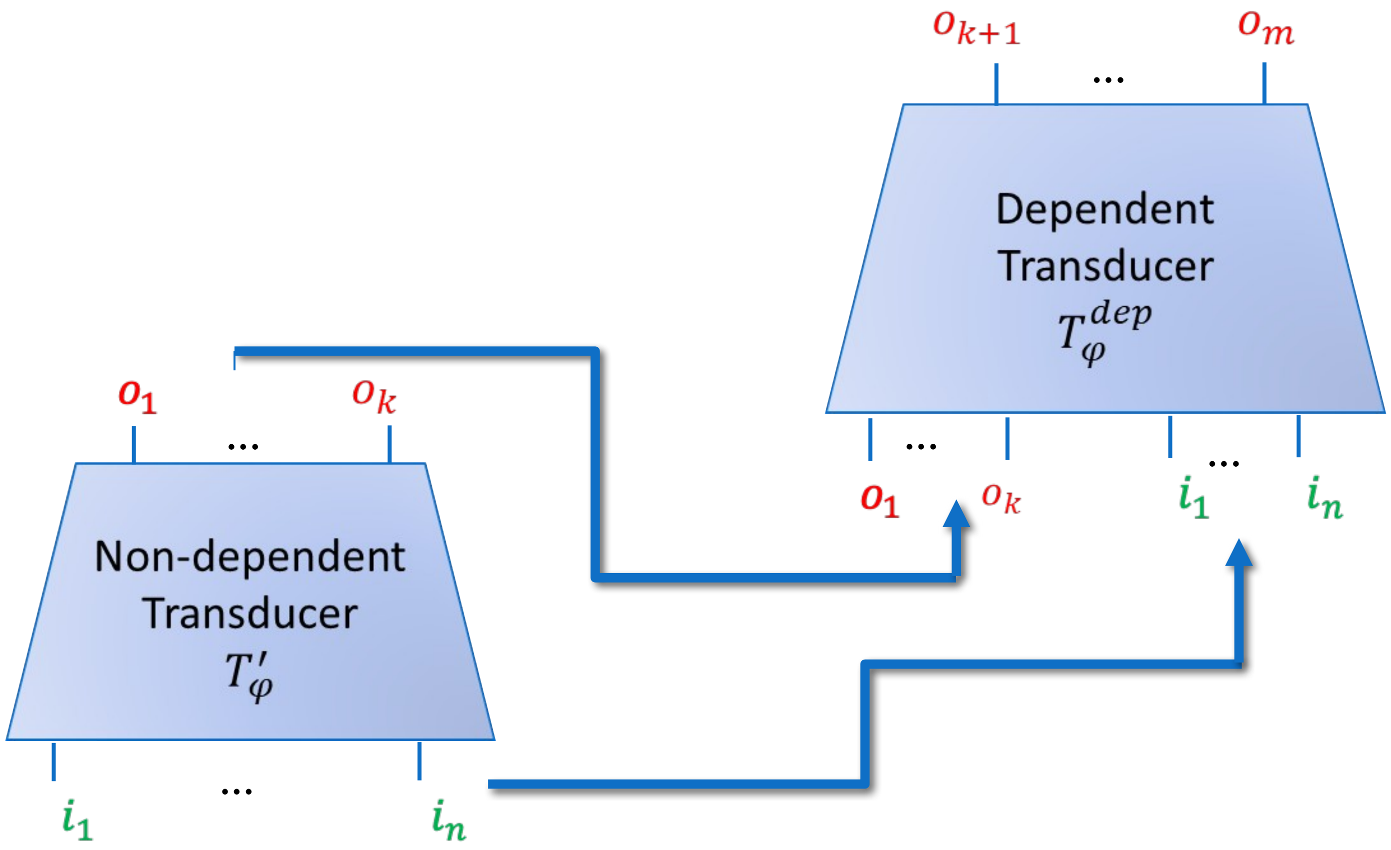


Step 6: Merge Transducers

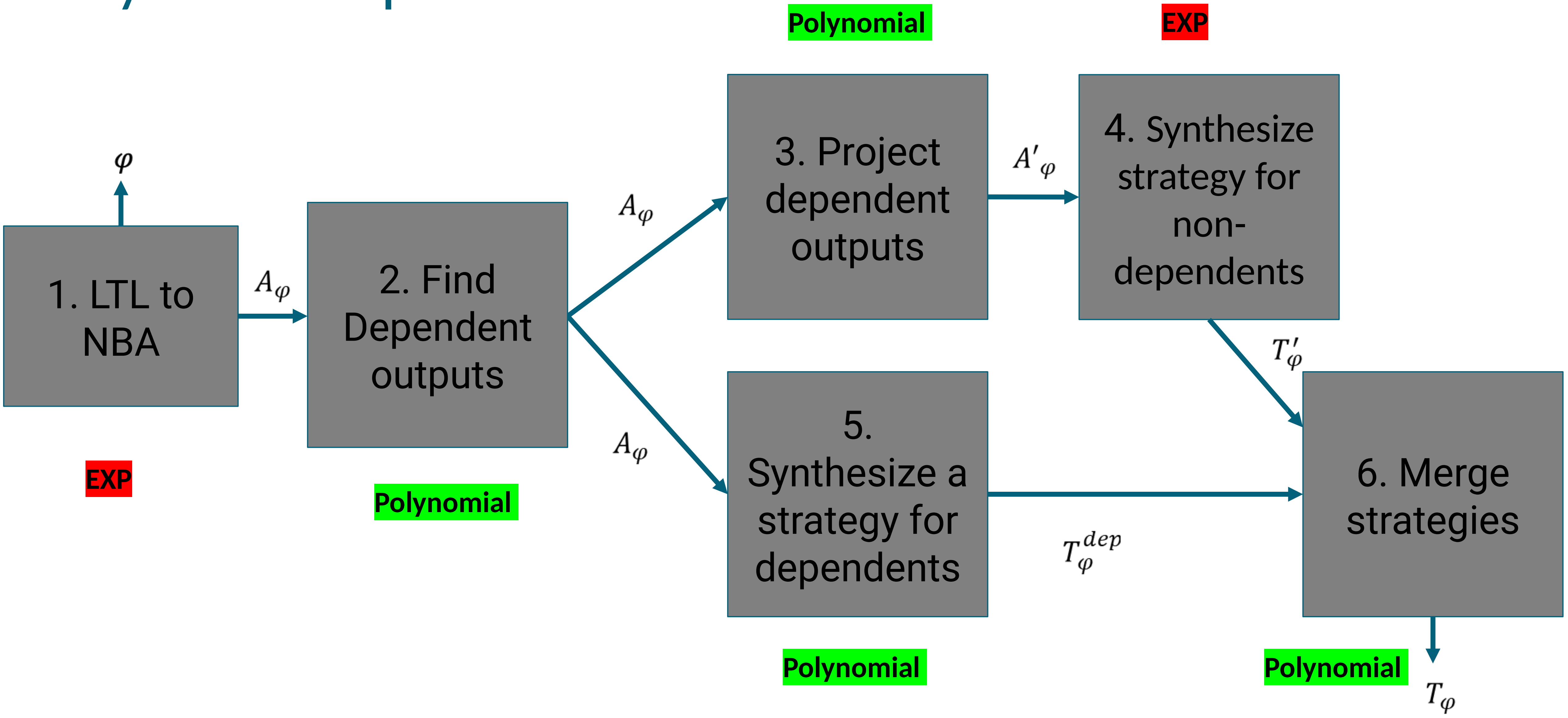


Step 6: Merge Transducers

- Merge is simply connecting outputs and inputs.



Synthesis Pipeline

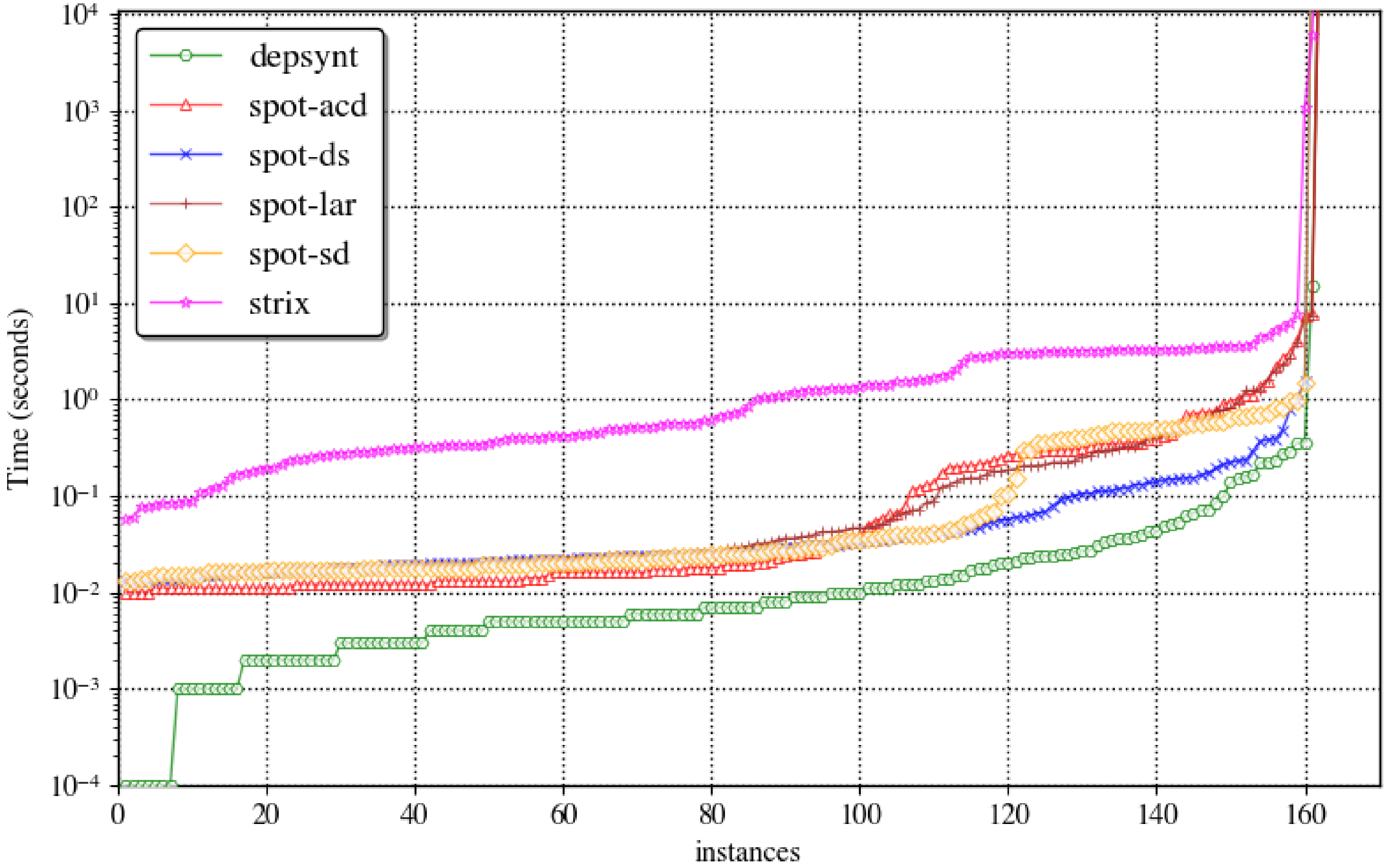


DepSynt Overview

- We implemented the synthesis pipeline in a tool called ***DepSynt***.
- DepSynt is developed in C++ using Spot [*Duret-Lutz. '14*] and our own implementation.
- Time for dependency-check is limited to 12 seconds.
  - Decided based on empirical results.
- We compared DepSynt with Ltlsynt (Spot) [Michaud,Colange, 2018] and Strix [Meyer, Sickert, Luttenberger 2018].

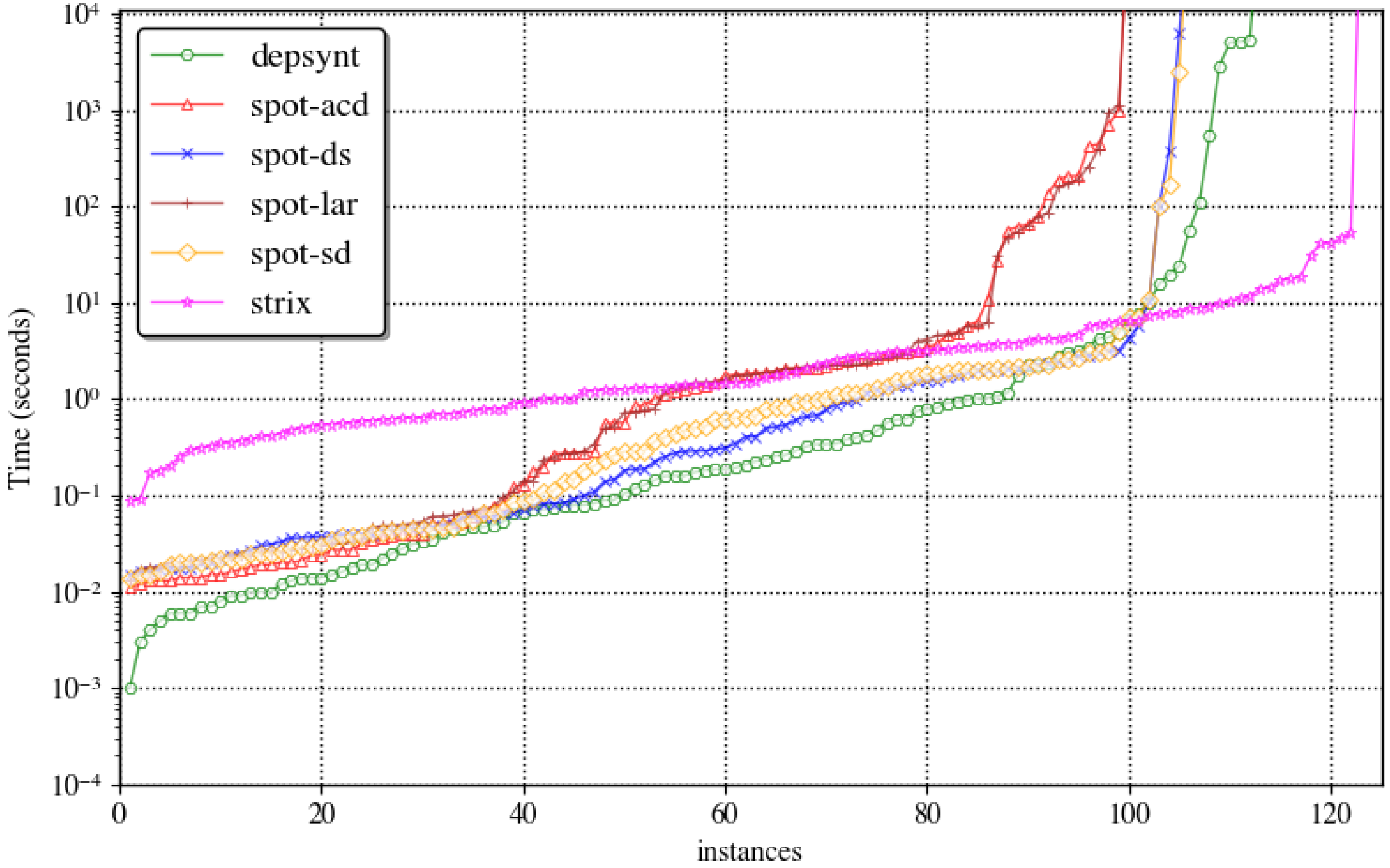
Non-dependent vars ≤ 3

- In benchmarks with at most 3 non-dependent variables.
- DepSynt outperforms state-of-the-art tools.



Non-dependent vars > 3

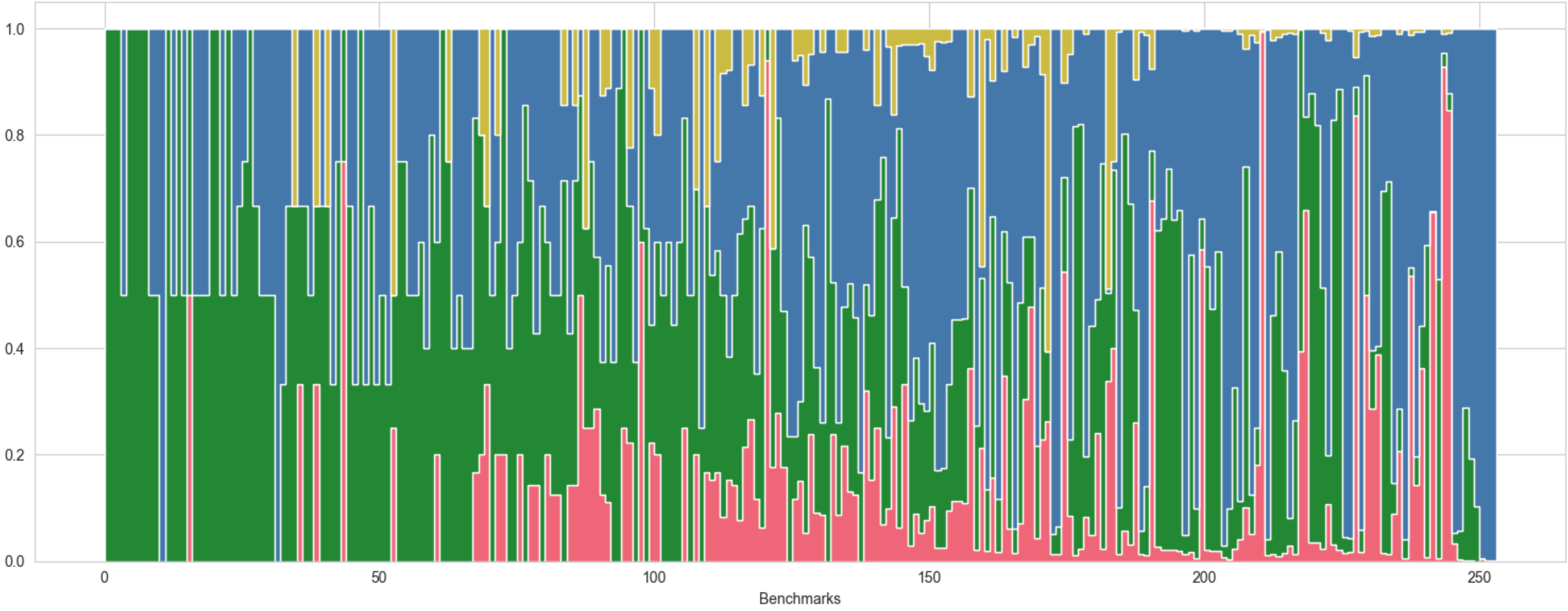
- In benchmarks with at more than 3 non-dependent variables.
- DepSynt is comparable with the other tools.



DepSynt - Time distribution

- How long DepSynt is spending on each phase - normalized.
- The benchmarks are sorted by total duration.

Search for dependent variables
Build NBA
Synthesis non-dependent variables
Synthesis dependent variables



Conclusion

- Formal definition of LTL dependency.
- Algorithm to find dependent variables.
- Framework that utilizes dependency for Reactive Synthesis.
- DepSynt confirms the dependency benefits.

Conclusion

- Formal definition of LTL dependency
- Algorithm to find dependent variables.
- Framework that utilizes dependency for Reactive Synthesis.
- DepSynt confirms the dependency benefits.
- Future work: exploring more general notions of dependency.