Example-Guided Synthesis of Relational Queries

Aalok Thakkar – Thesis Proposal

December 14, 2022







Outline

1. Context
2. Timeline
3. Synthesis of Conjunctive Queries
4. Decidability and Complexity Results
5. Extension 1: Union
6. Extension 2: Comparison Predicates
7. Extension 3: Recursion
8. Conclusion and Future Work

Example-Guided Synthesis
of Relational Queries

Example-Guided Synthesis of Relational Queries

declarative logic programs

SQL

Datalog

Cypher

SPARQL

Example-Guided Synthesis of Relational Queries

declarative logic programs

PQL

Prolog

LogiQL

CodeQL

Example-Guided Synthesis of Relational Queries

declarative logic programs

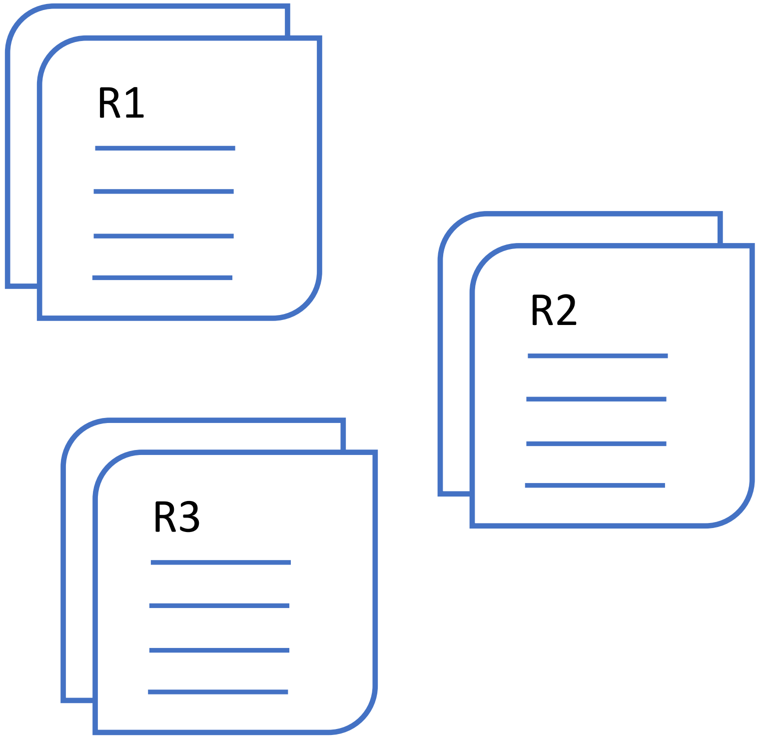
Knowledge Discovery

Program Analysis

Database Querying

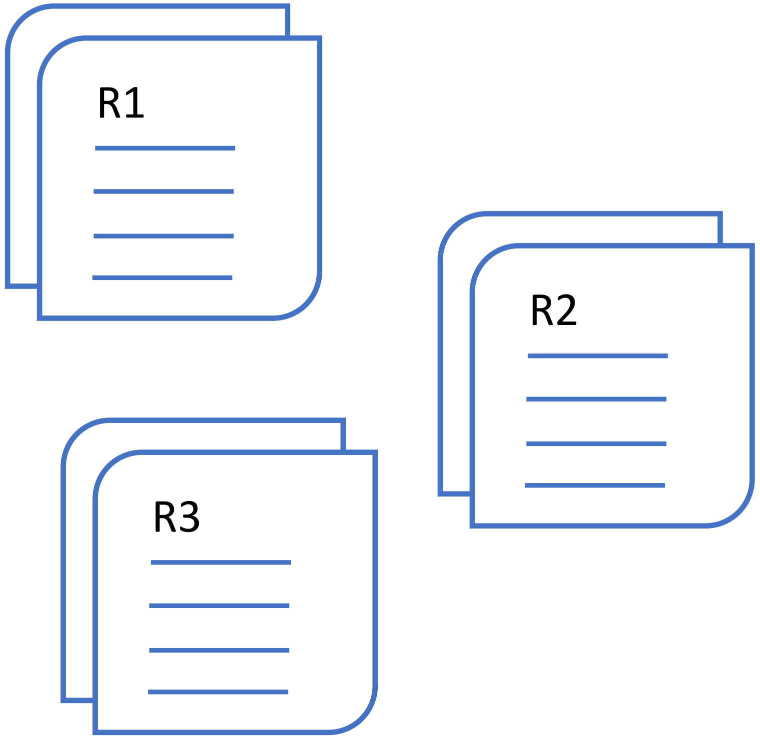
Example-Guided Synthesis of Relational Queries

Query Synthesis Problem



Input Tables

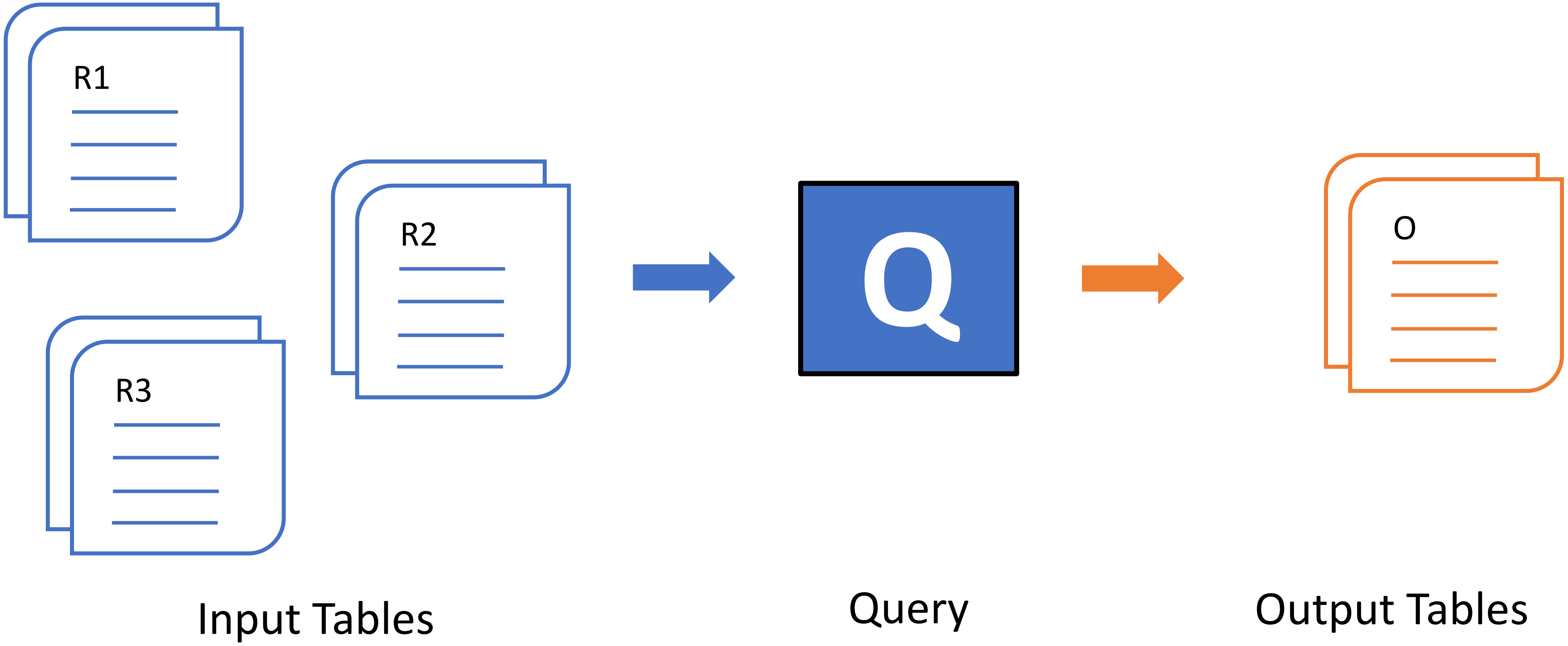
Query Synthesis Problem



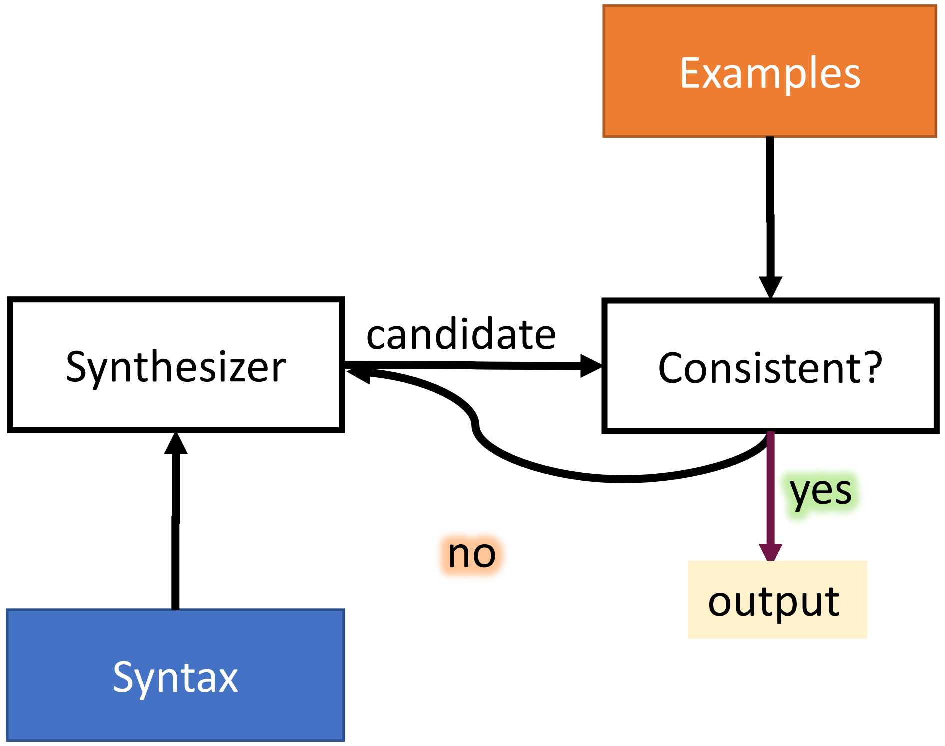
Input Tables

Output Tables

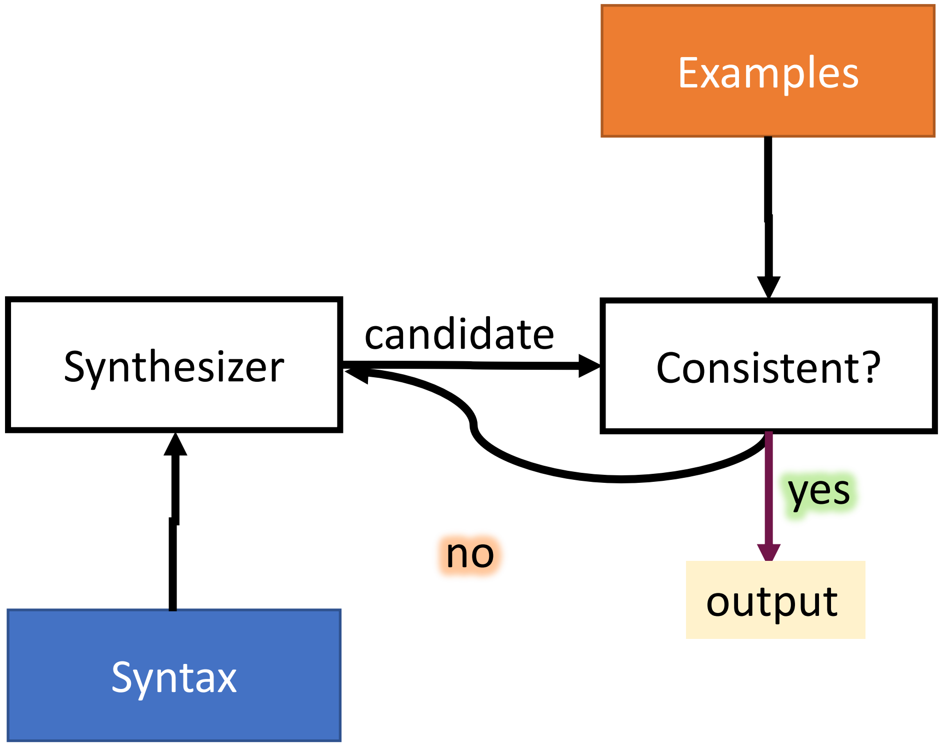
Query Synthesis Problem



Syntax-guided Techniques



Syntax-guided Techniques

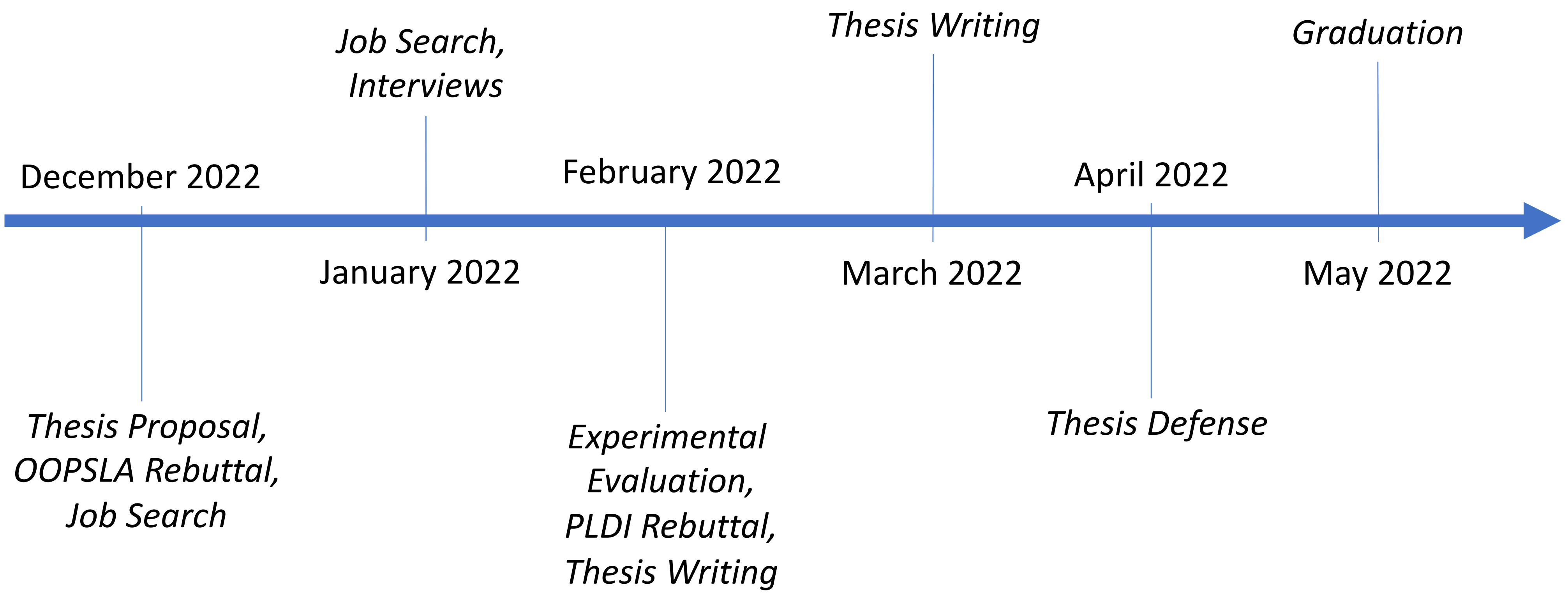


Scythe

Prosynth

ILASP

Proposed Timeline



PLDI 2021

Example-Guided Synthesis of Relational Queries

Aalok Thakkar
University of Pennsylvania
Philadelphia, USA
athakkar@cis.upenn.edu

Aaditya Naik
University of Pennsylvania
Philadelphia, USA
asnaik@cis.upenn.edu

Nathaniel Sands
University of Southern California
Los Angeles, USA
njsands@usc.edu

Rajeev Alur
University of Pennsylvania
Philadelphia, USA
alur@cis.upenn.edu

Mayur Naik
University of Pennsylvania
Philadelphia, USA
mhnaik@cis.upenn.edu

Mukund Raghothaman
University of Southern California
Los Angeles, USA
raghotha@usc.edu

Abstract

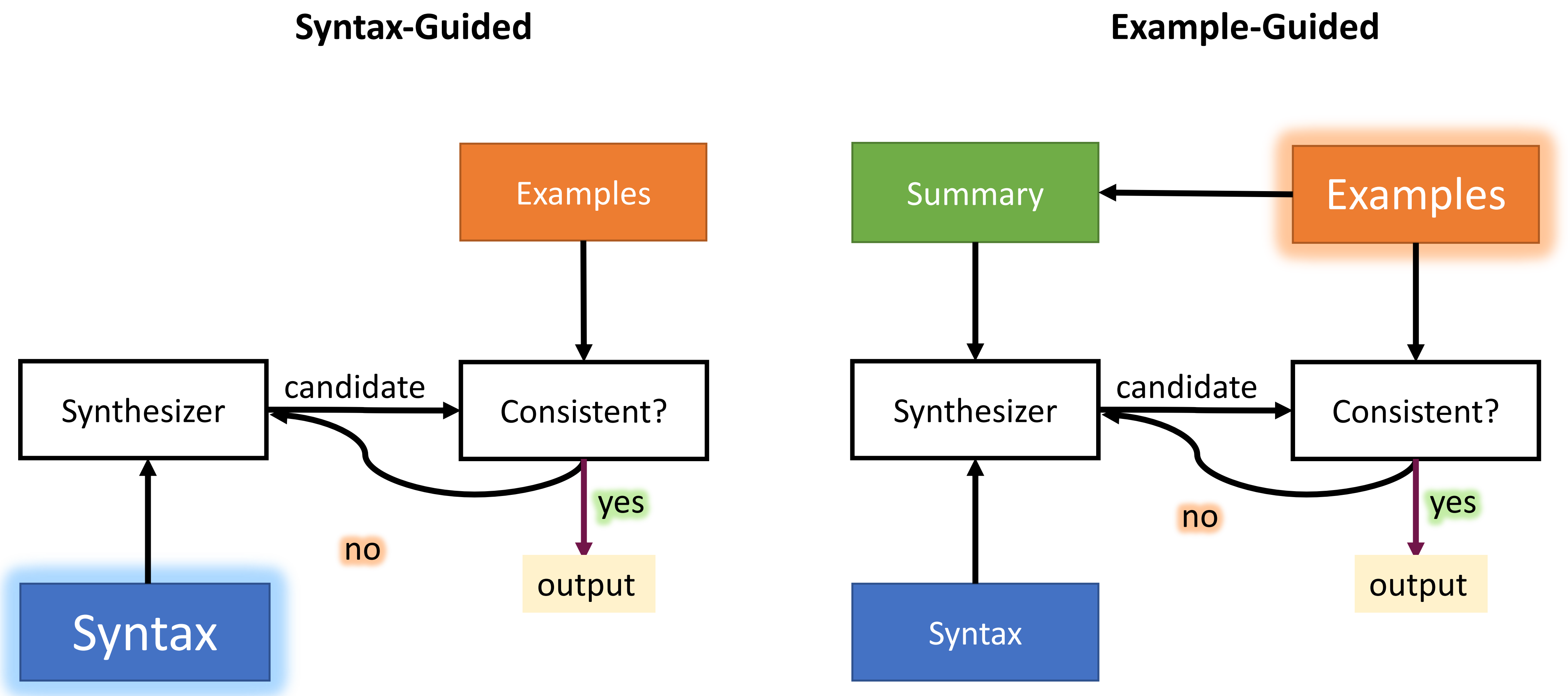
Program synthesis tasks are commonly specified via input-output examples. Existing enumerative techniques for such tasks are primarily guided by program syntax and only make indirect use of the examples. We identify a class of synthesis algorithms for programming-by-examples, which we call Example-Guided Synthesis (EGS), that exploits latent structure in the provided examples while generating candidate programs. We present an instance of EGS for the synthesis of relational queries and evaluate it on 86 tasks from three application domains: knowledge discovery, program analy-

1 Introduction

Program synthesis aims to automatically synthesize a program that meets user intent. While the user intent is classically described as a correctness specification, synthesizing programs from input-output examples has gained much traction, as evidenced by the many applications of programming-by-example and programming-by-demonstration, such as spreadsheet programming [25], relational query synthesis [51, 57], and data wrangling [19, 33]. Nevertheless, their scalability remains an important challenge, and often hinders their application in the field [5].

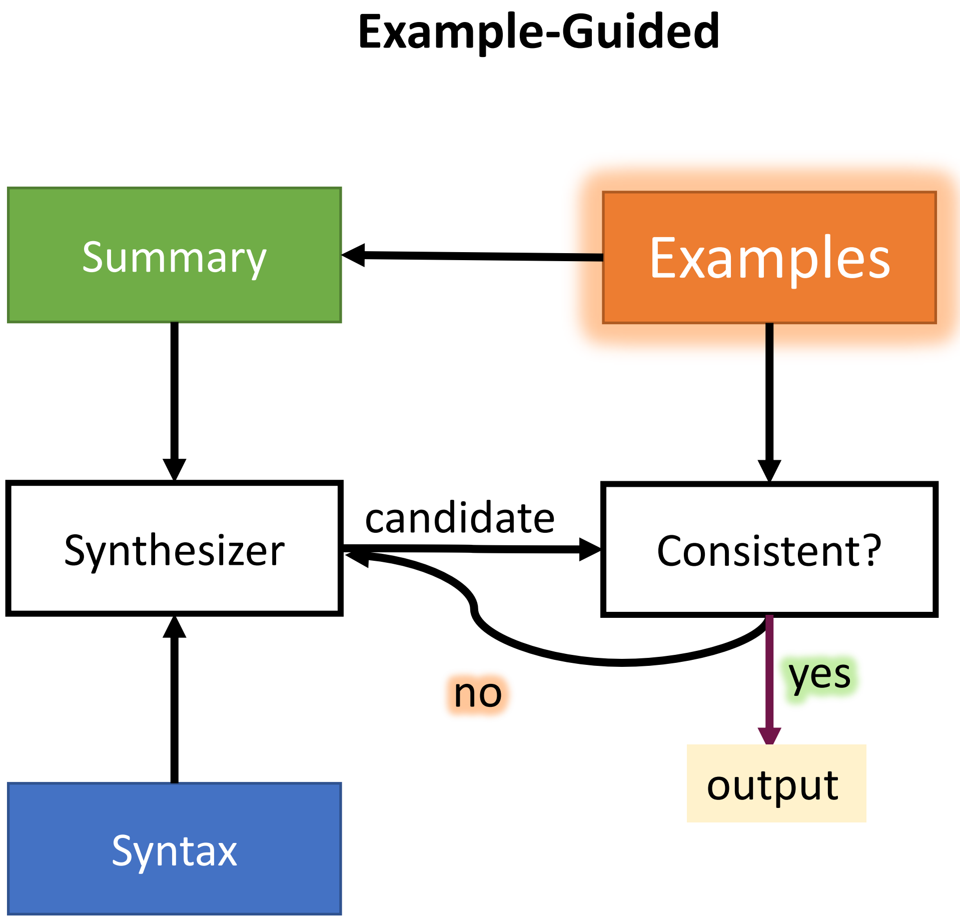
Example-Guided Synthesis of Relational Queries

Example-guided Synthesis



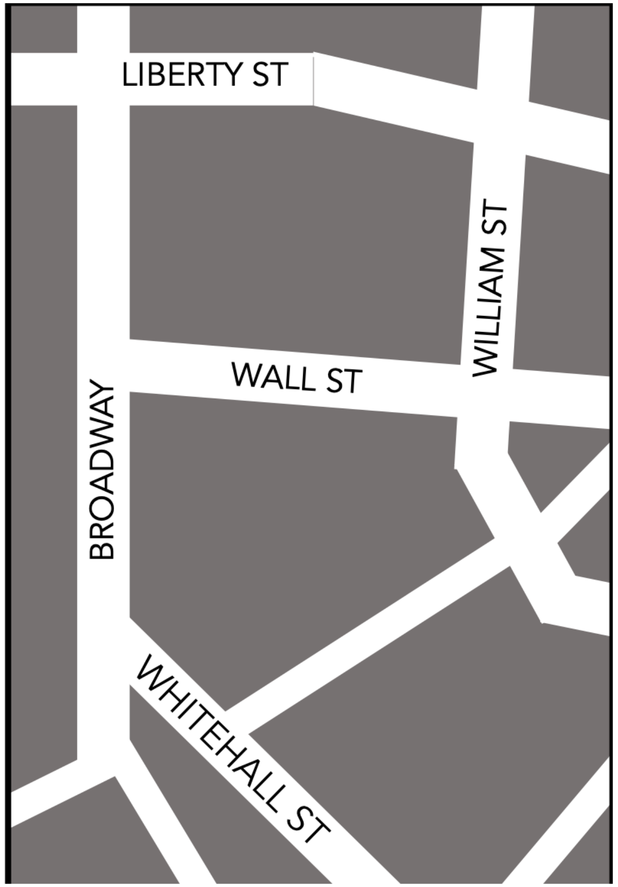
Example-guided Synthesis

1. Examples cannot be replaced by an evaluation oracle
2. Uses the latent *structure* of examples to generate the candidate programs
3. Outperforms syntax-guided techniques for relational queries



Example-Guided Synthesis of Relational Queries

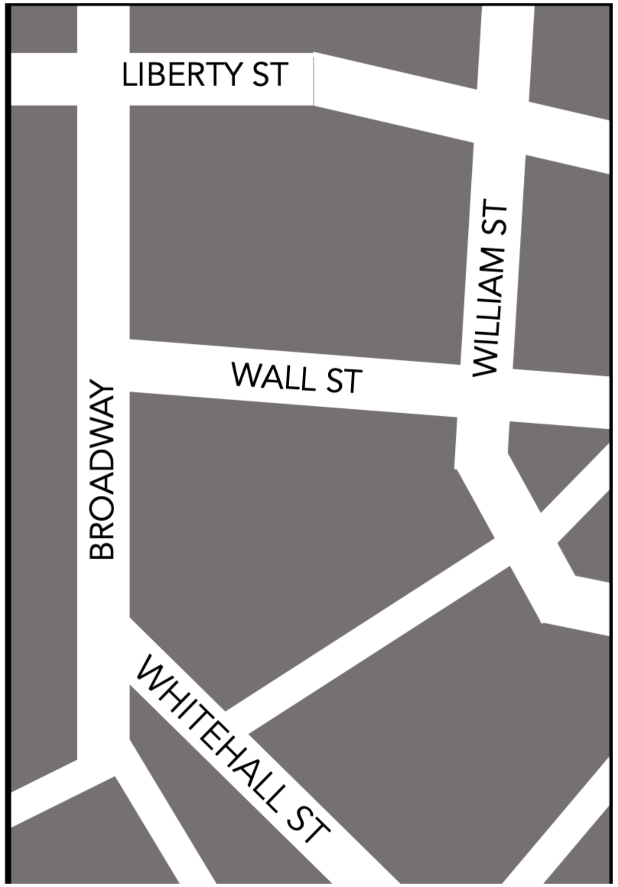
An Example



GreenSignal
Broadway
Liberty St
William St
Whitehall St

HasTraffic
Broadway
Wall St
William St
Whitehall St

An Example



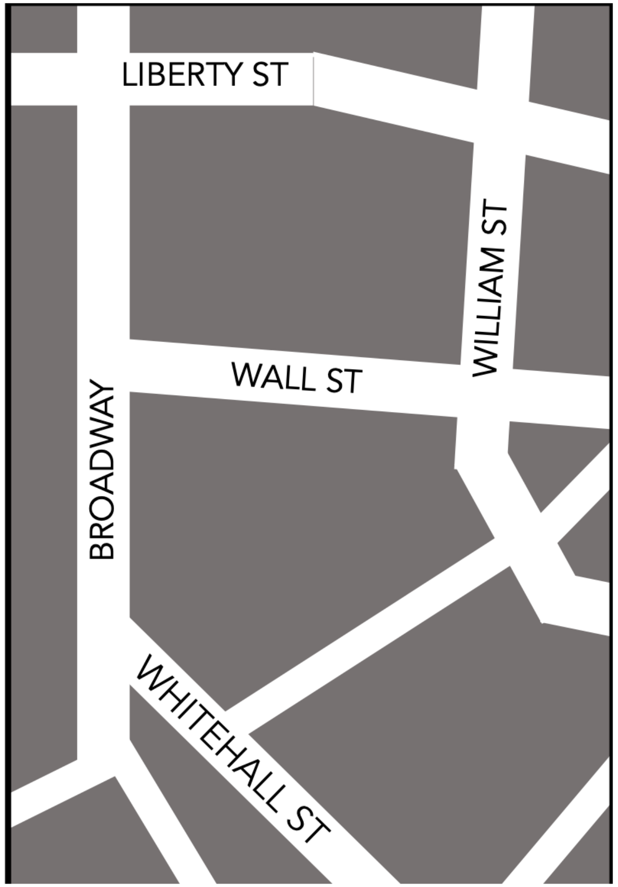
GreenSignal
Broadway
Liberty St
William St
Whitehall St

HasTraffic
Broadway
Wall St
William St
Whitehall St

→

Crashes
Broadway
Whitehall St

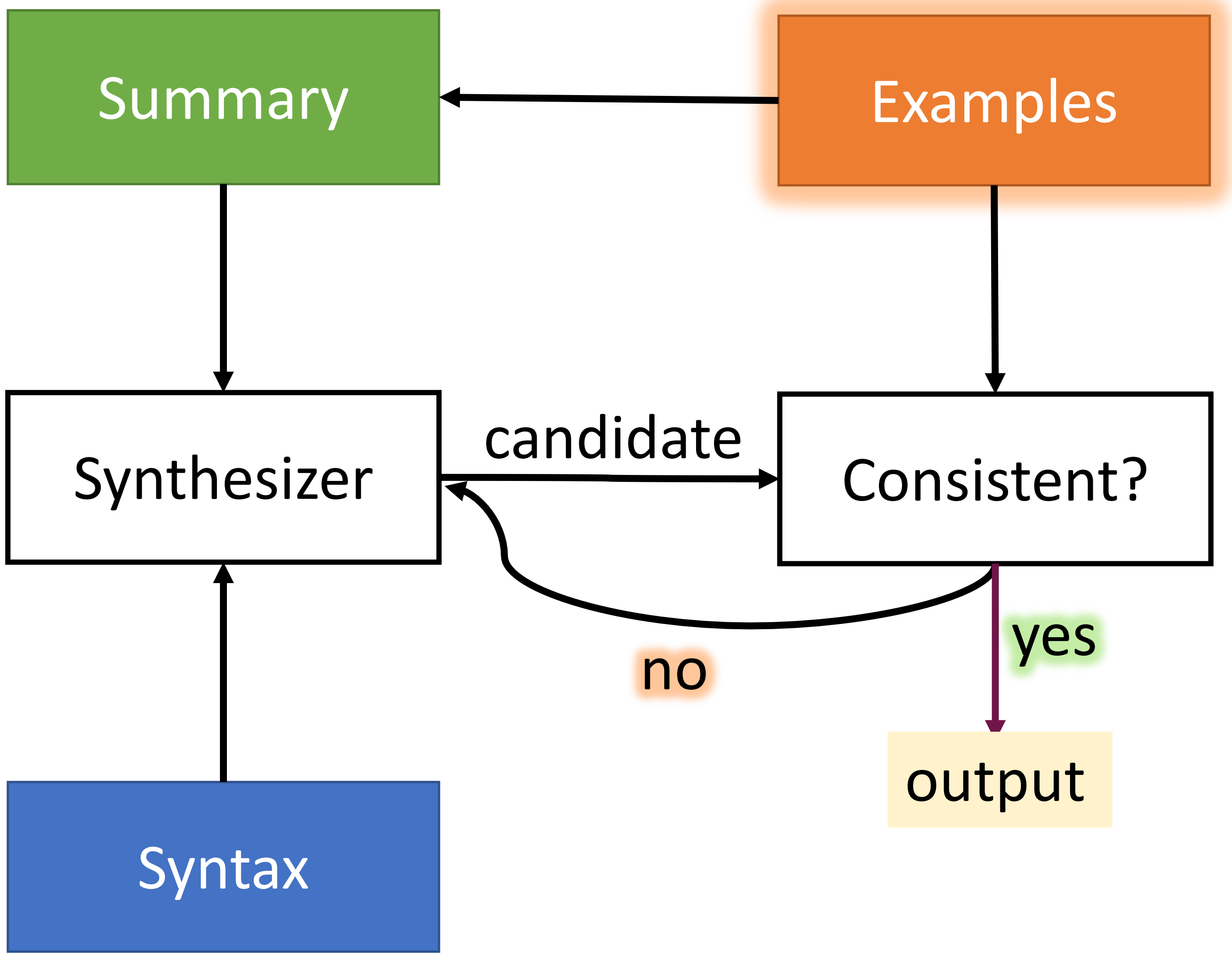
An Example

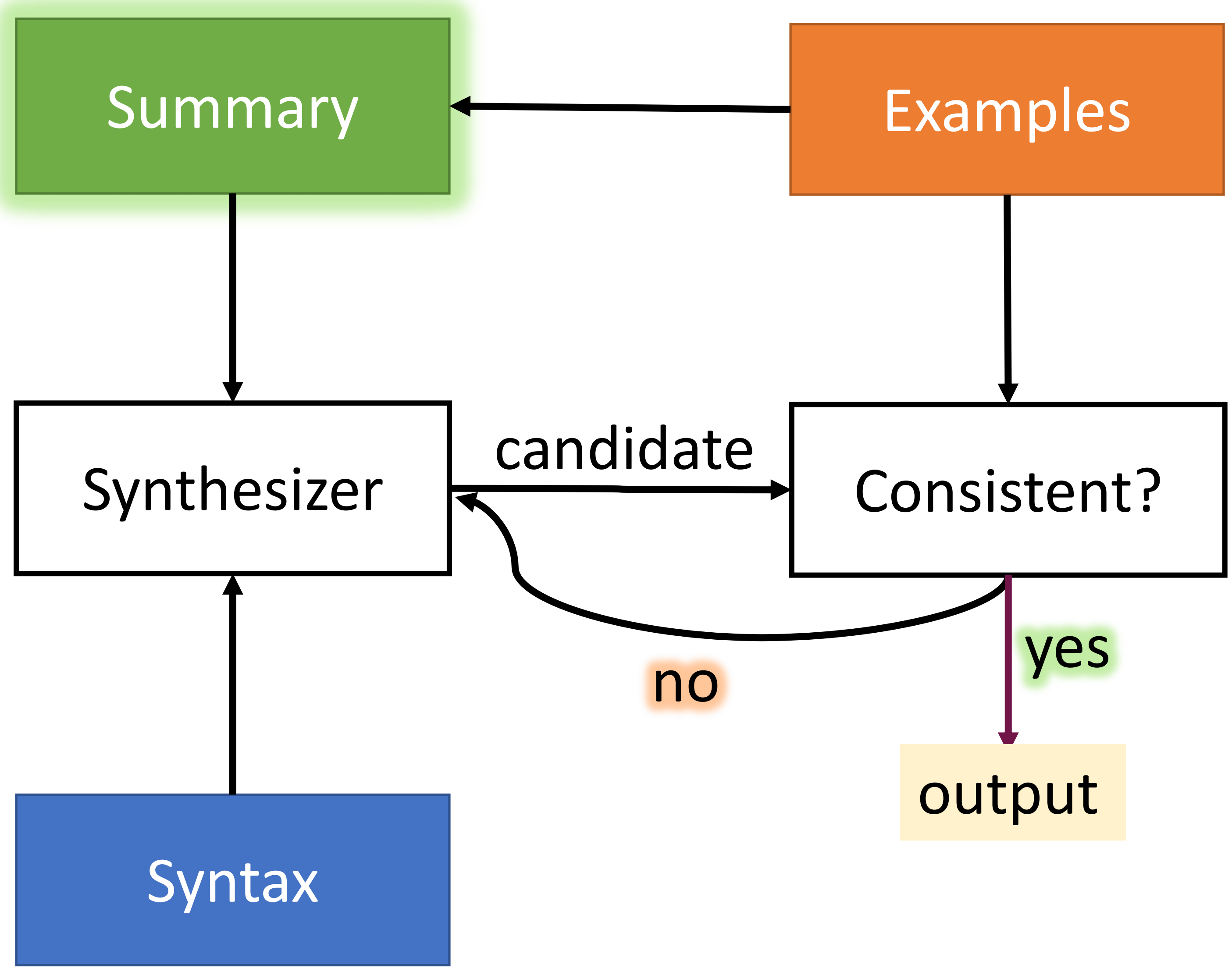


GreenSignal
Broadway
Liberty St
William St
Whitehall St

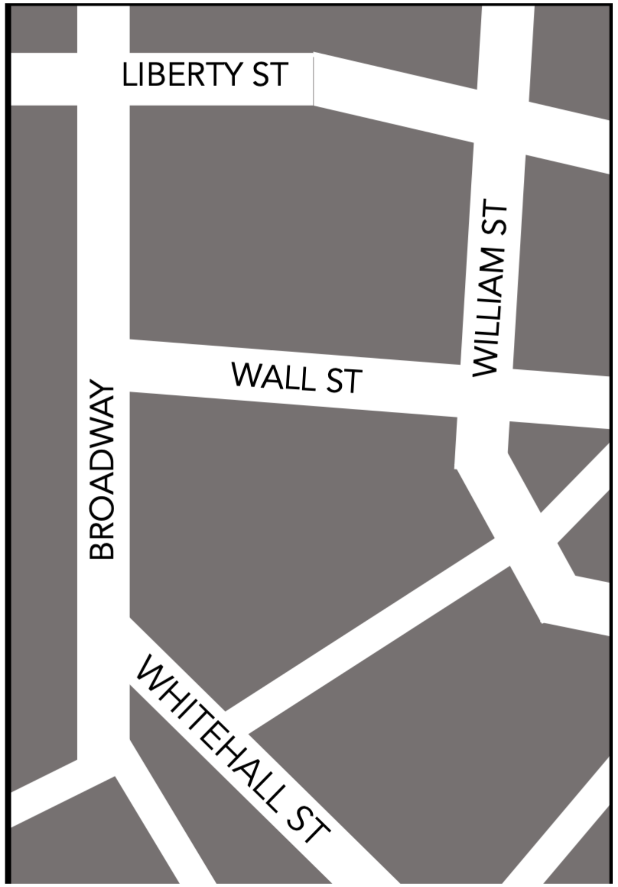
HasTraffic
Broadway
Wall St
William St
Whitehall St

```
Crashes(x) :— HasTraffic(x).
Crashes(x) :— GreenSignal(x).
Crashes(x) :— Intersects(x, y).
Crashes(x) :— Intersects(y, x).
Crashes(x) :— HasTraffic(x), GreenSignal(x).
Crashes(x) :— HasTraffic(x), Intersects(x, y).
Crashes(x) :— HasTraffic(x), Intersects(y, x).
Crashes(x) :— GreenSignal(x), Intersects(x, y).
Crashes(x) :— GreenSignal(x), Intersects(y, x).
Crashes(x) :— Intersects(x, y), Intersects(y, x).
Crashes(x) :— HasTraffic(x), GreenSignal(x),
                  Intersects(x, y).
Crashes(x) :— HasTraffic(x), GreenSignal(y),
                  Intersects(y, x).
…
```


Summary
Examples
Synthesizer
candidate
Consistent?
no
yes
output
Syntax

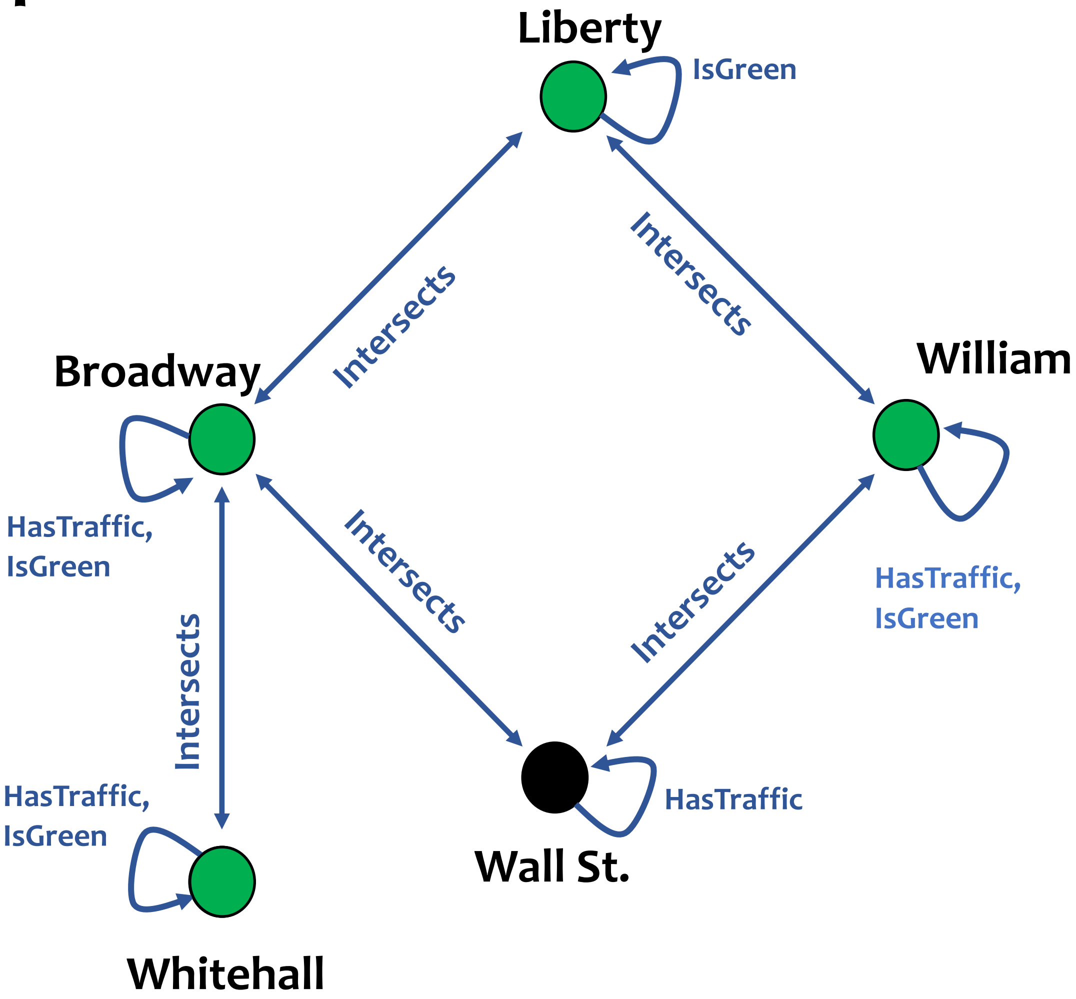

Summary
Examples
candidate
Synthesizer
Consistent?
yes
no
output
Syntax

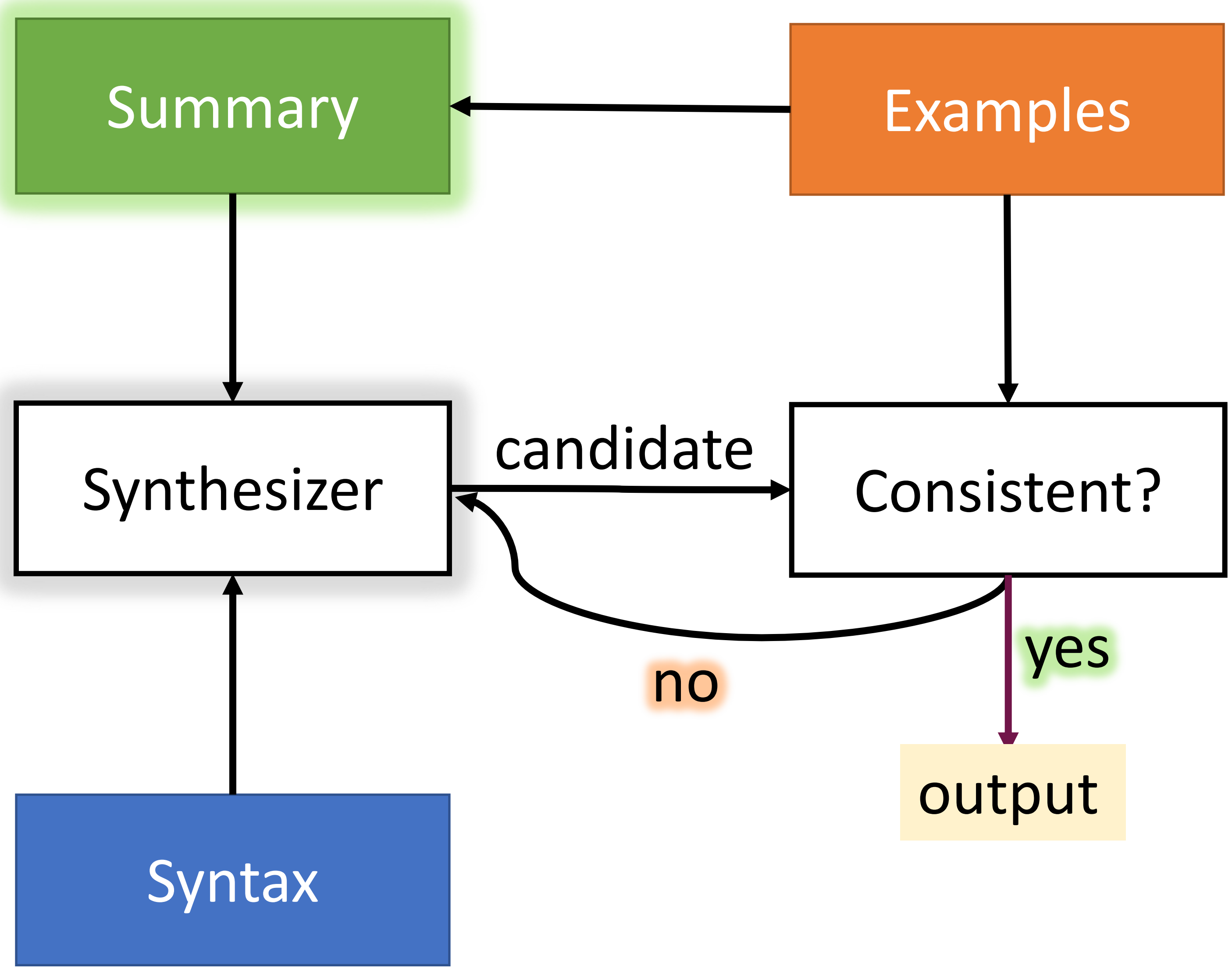
An Example



GreenSignal
Broadway
Liberty St
William St
Whitehall St

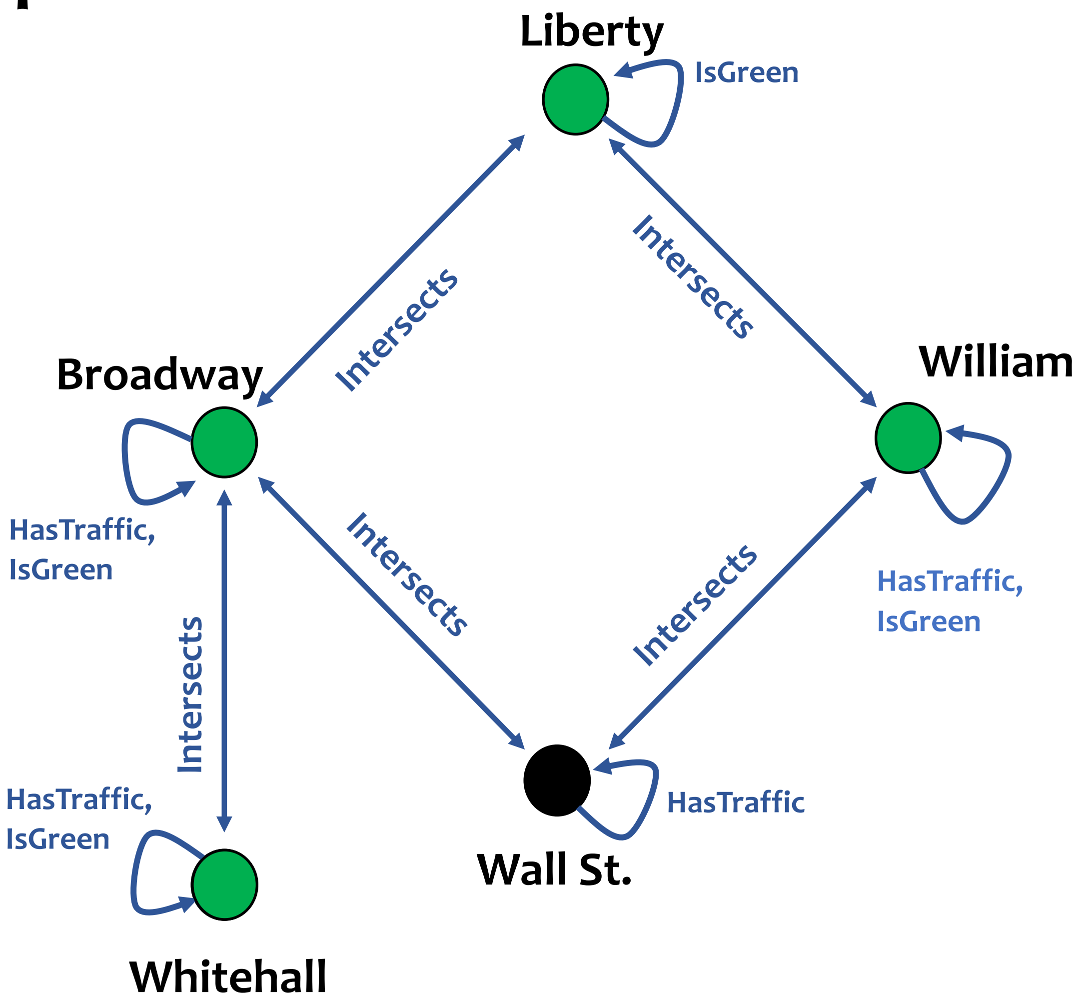
HasTraffic
Broadway
Wall St
William St
Whitehall St




Summary
Examples
Synthesizer
candidate
Consistent?
yes
no
output
Syntax

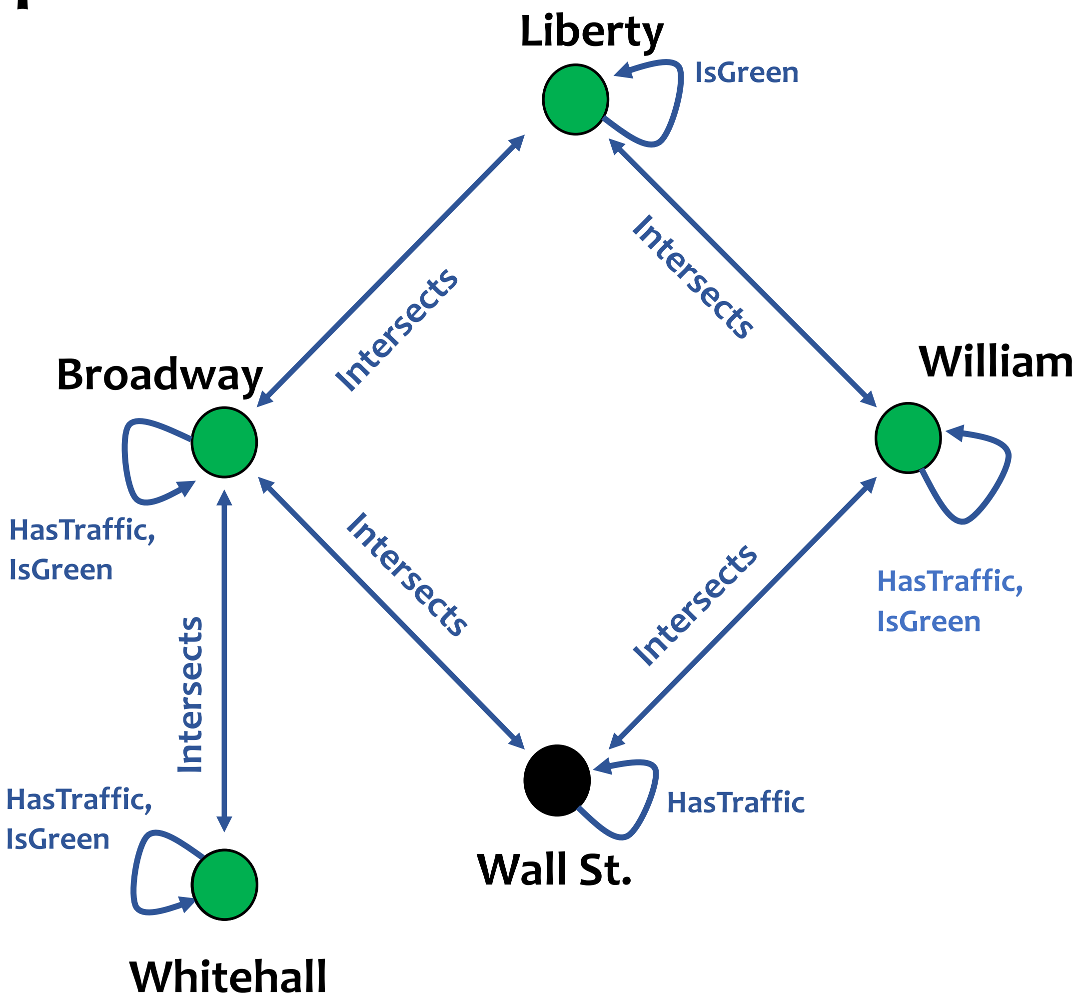
An Example

Crashes
Broadway
Whitehall St



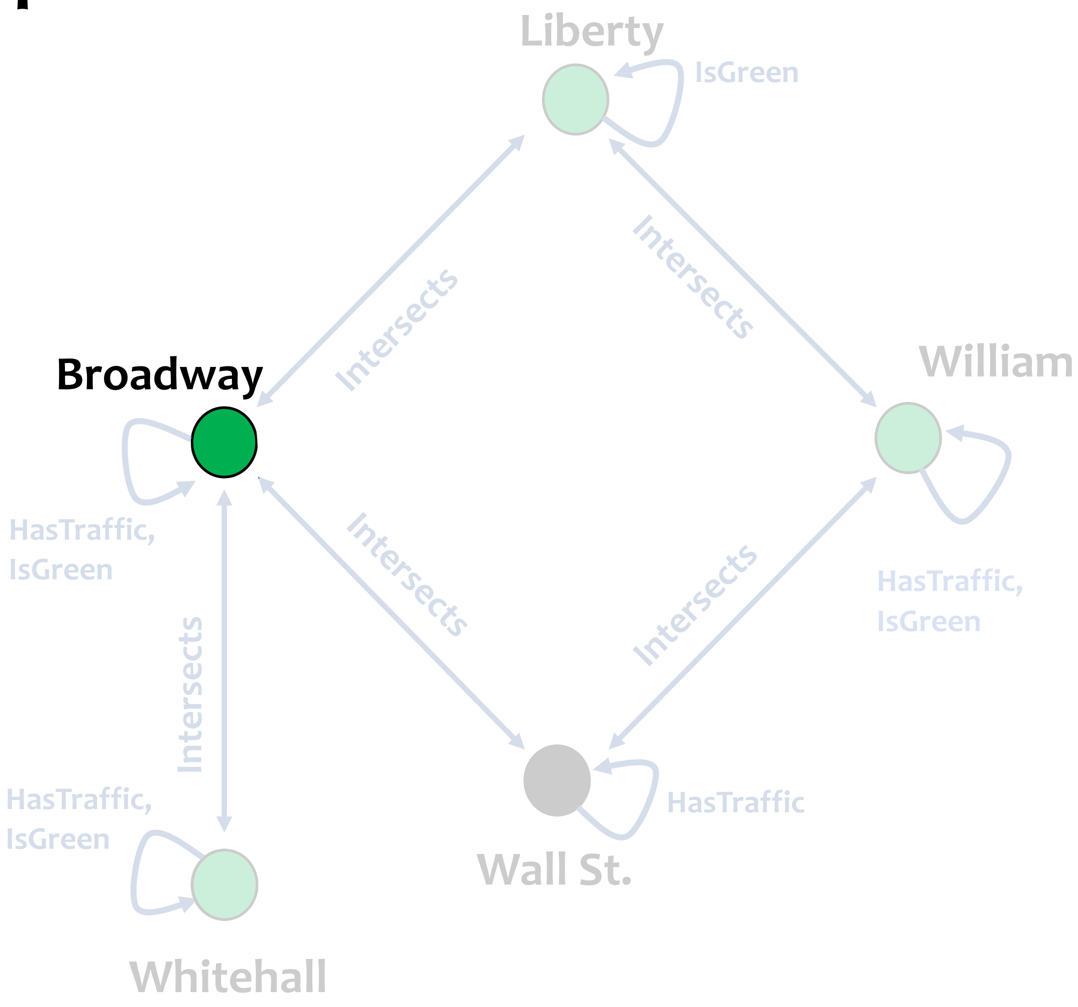
An Example

Crashes
Broadway
Whitehall St



An Example

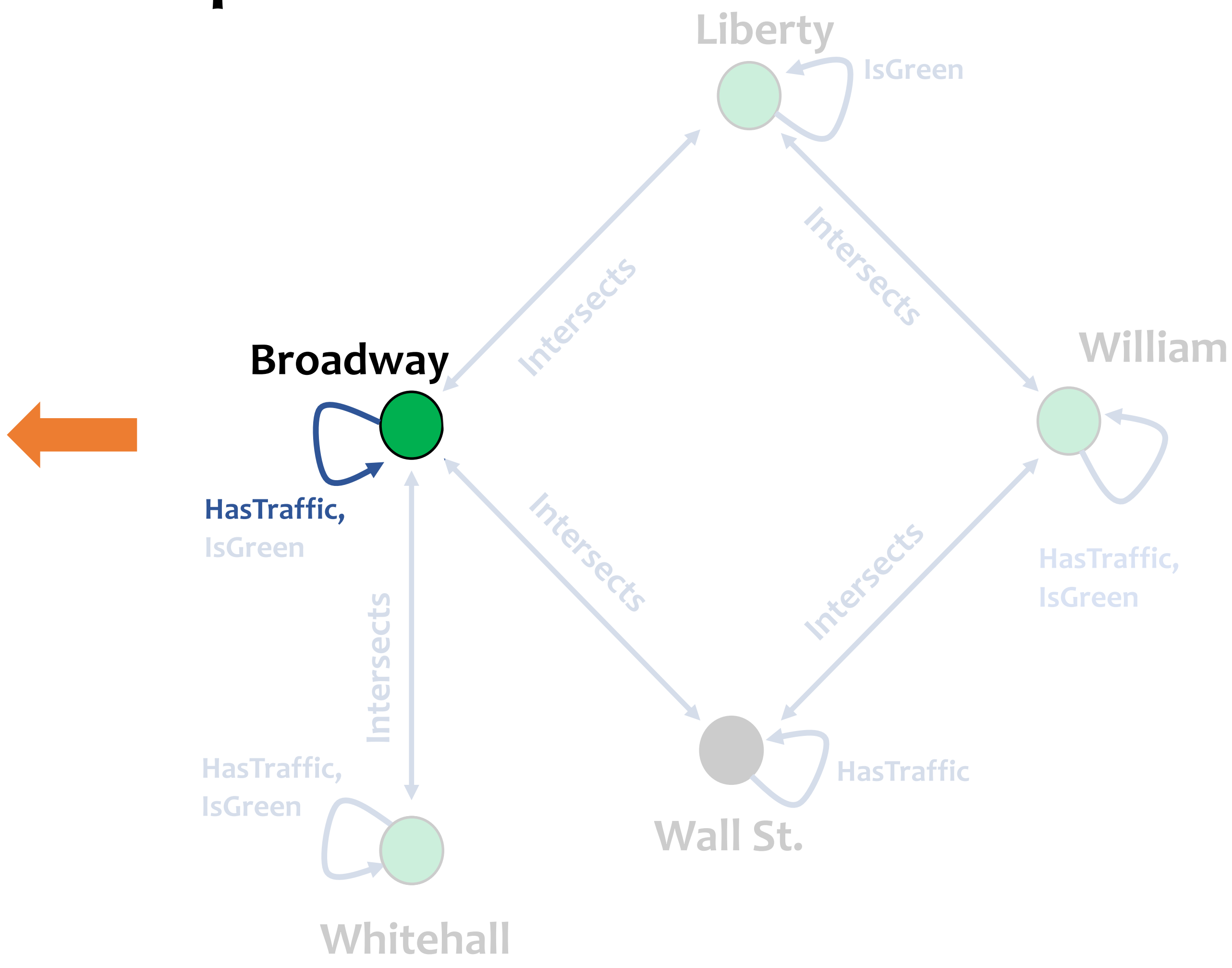
Crashes
Broadway
Whitehall St



An Example

Crashes
Broadway
Whitehall St

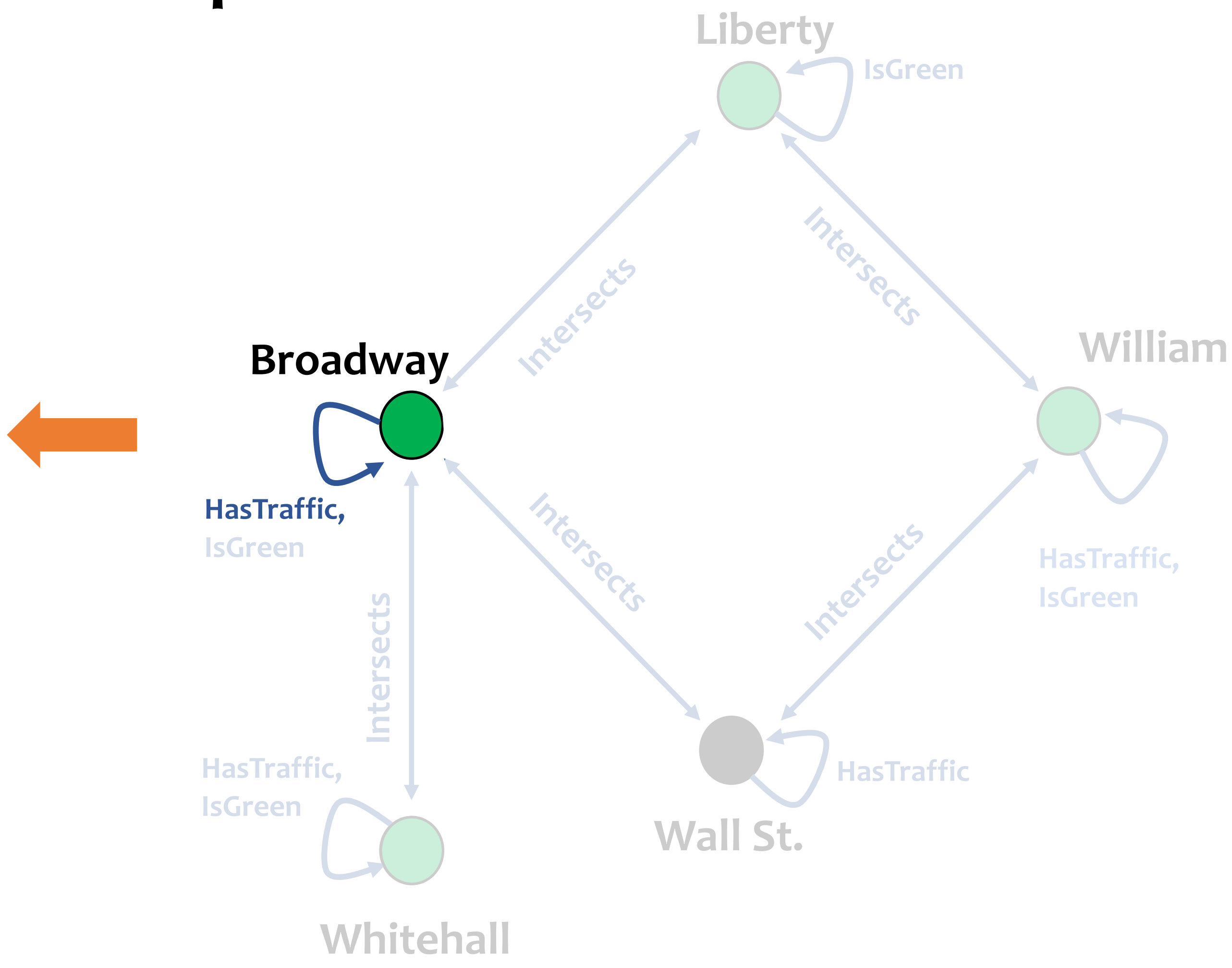
Crashes(Broadway) ← HasTraffic(Broadway).



An Example

Crashes
Broadway
Whitehall St

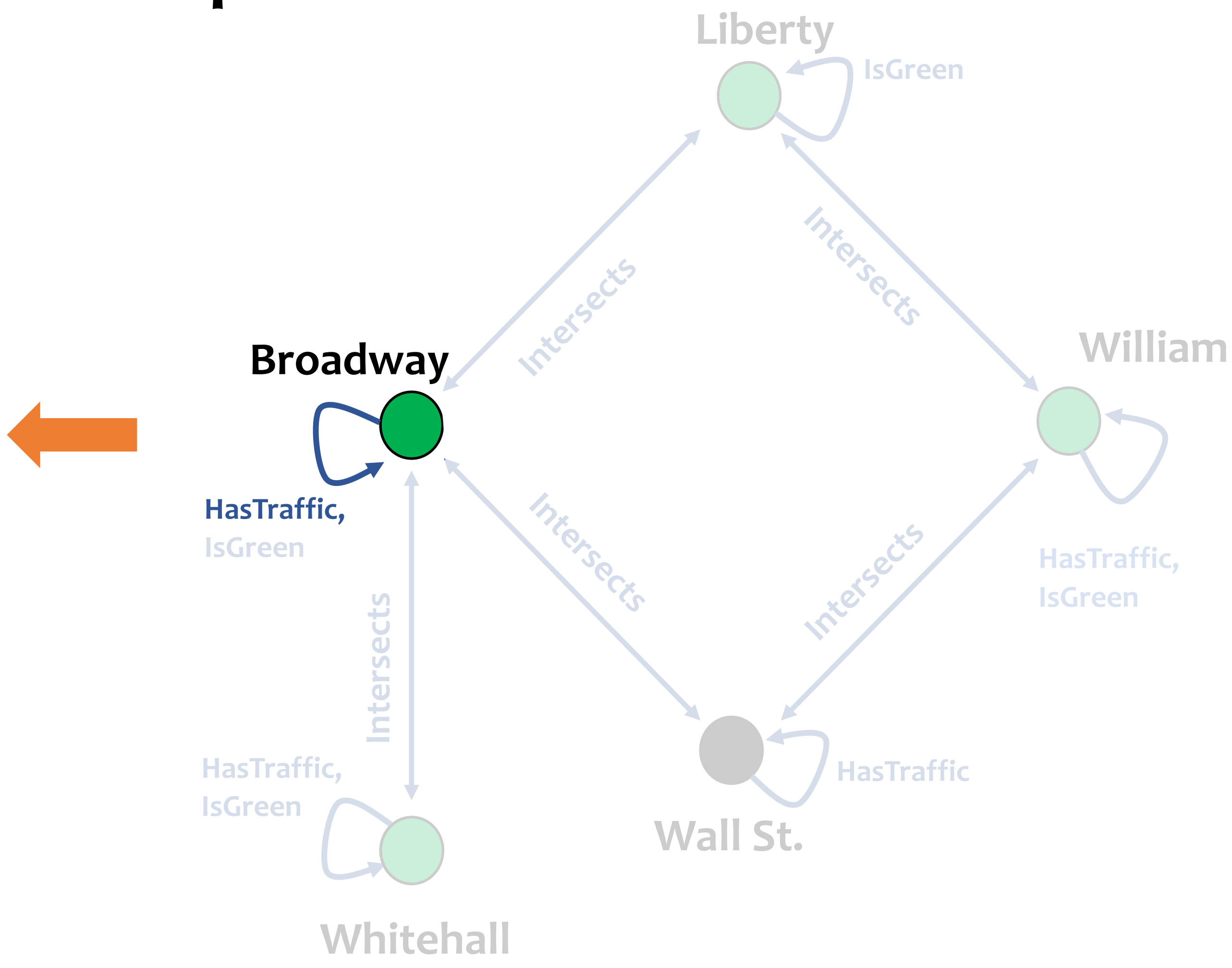
Crashes(Broadway) :– HasTraffic(Broadway).

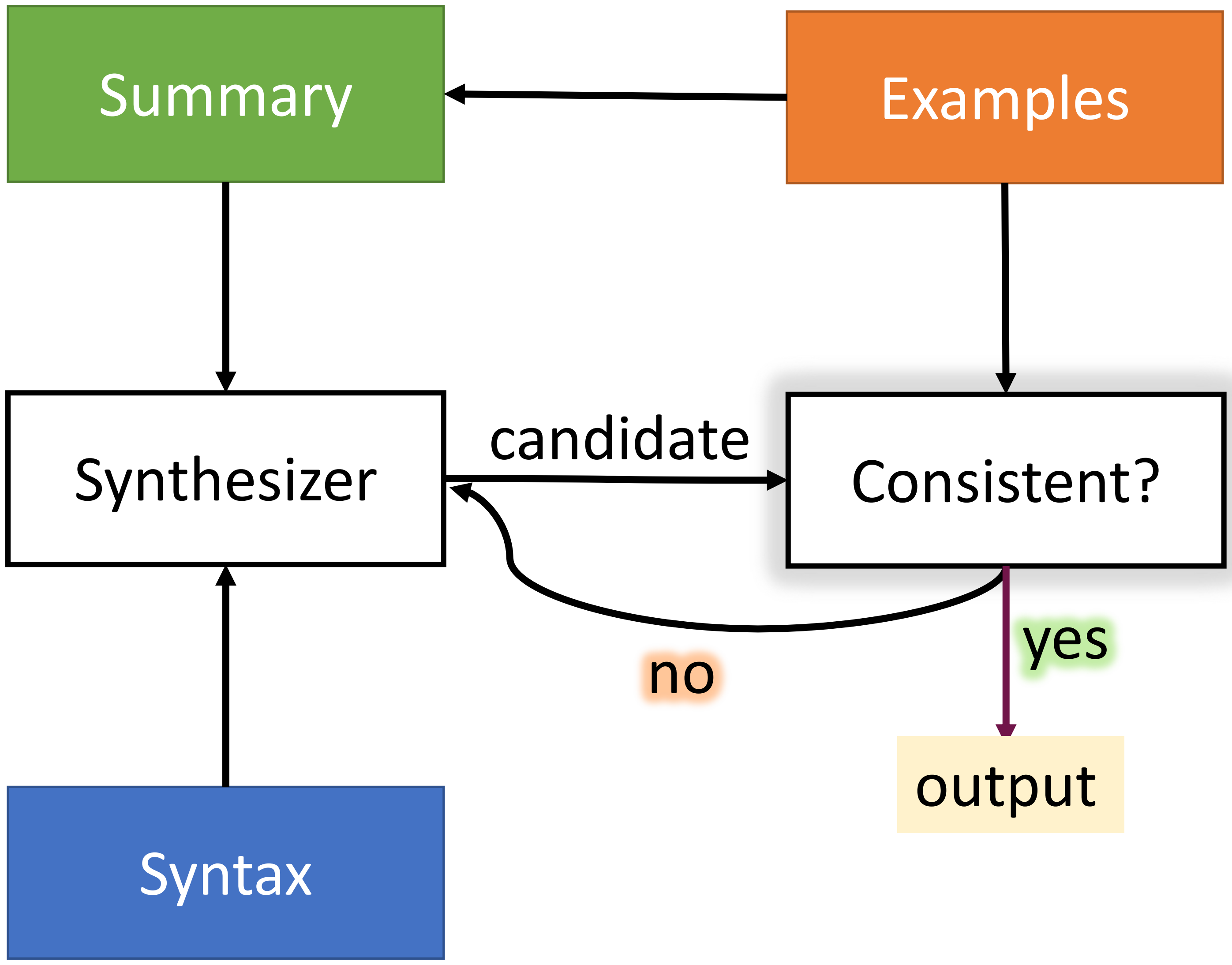


An Example

Crashes
Broadway
Whitehall St

Crashes(x) :– HasTraffic(x).

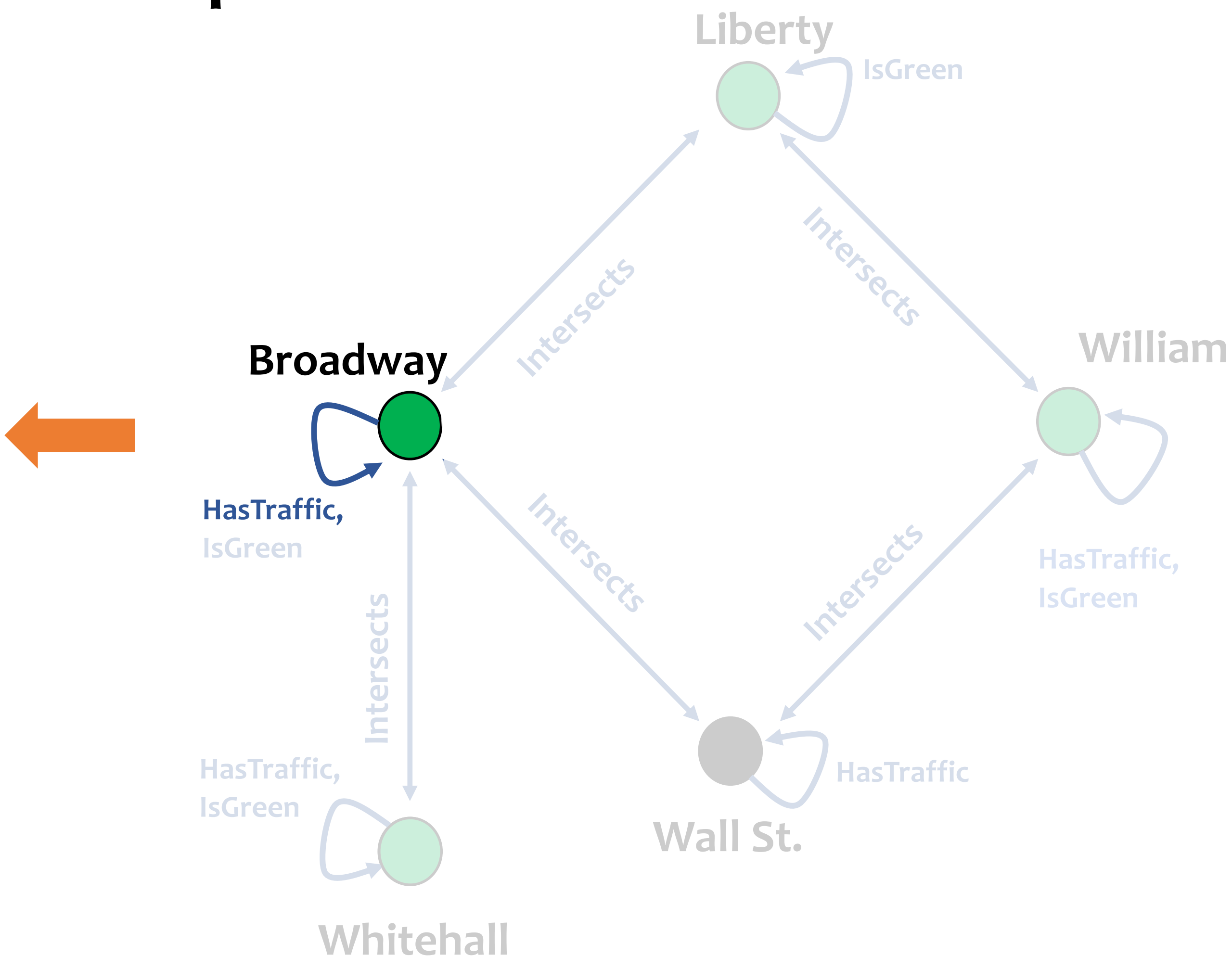



Summary
Examples
Synthesizer
candidate
Consistent?
yes
no
output
Syntax

An Example

Crashes
Broadway
Whitehall St

Crashes(x) :– HasTraffic(x).



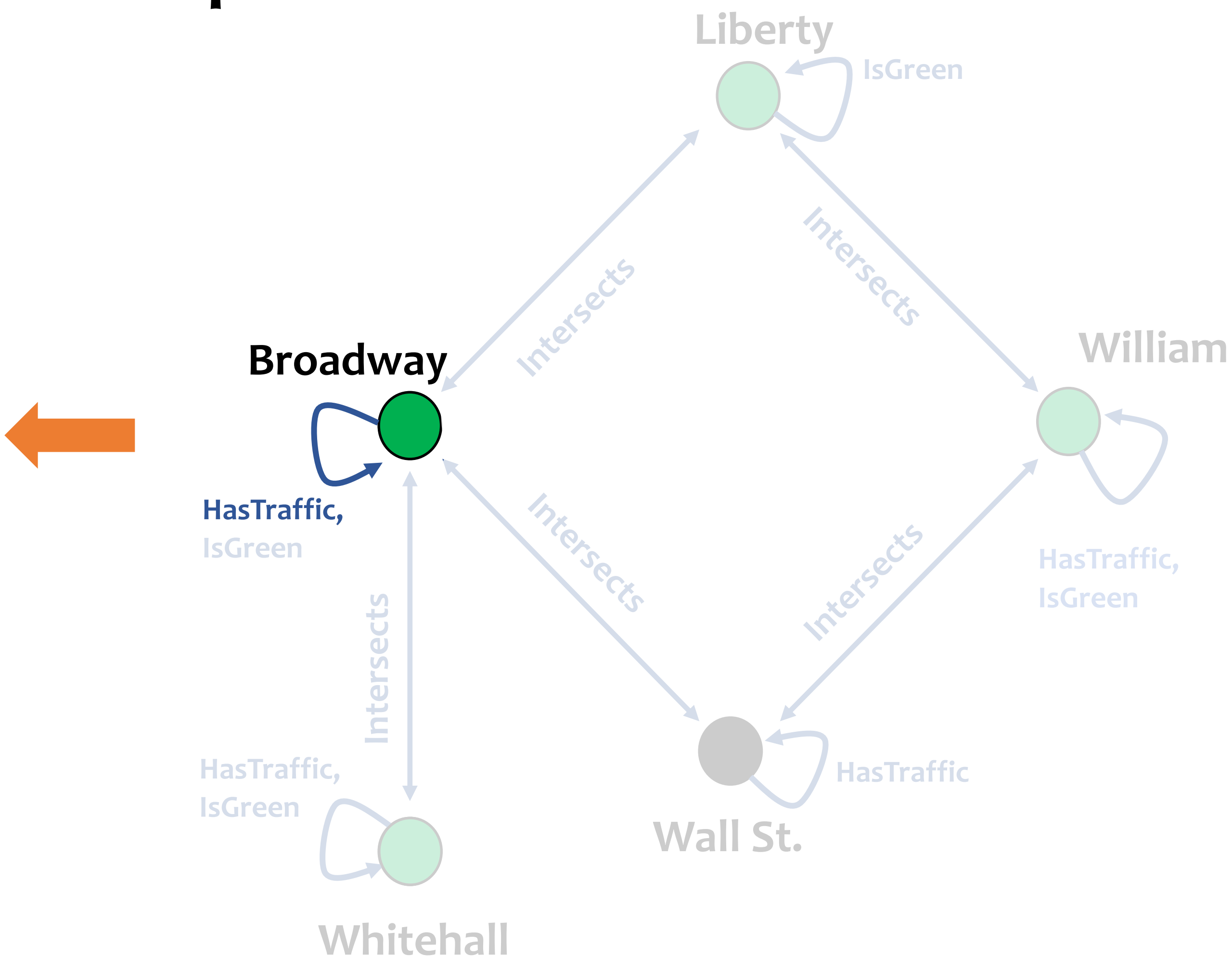
Crashes
Broadway
Wall St
William St
Whitehall St

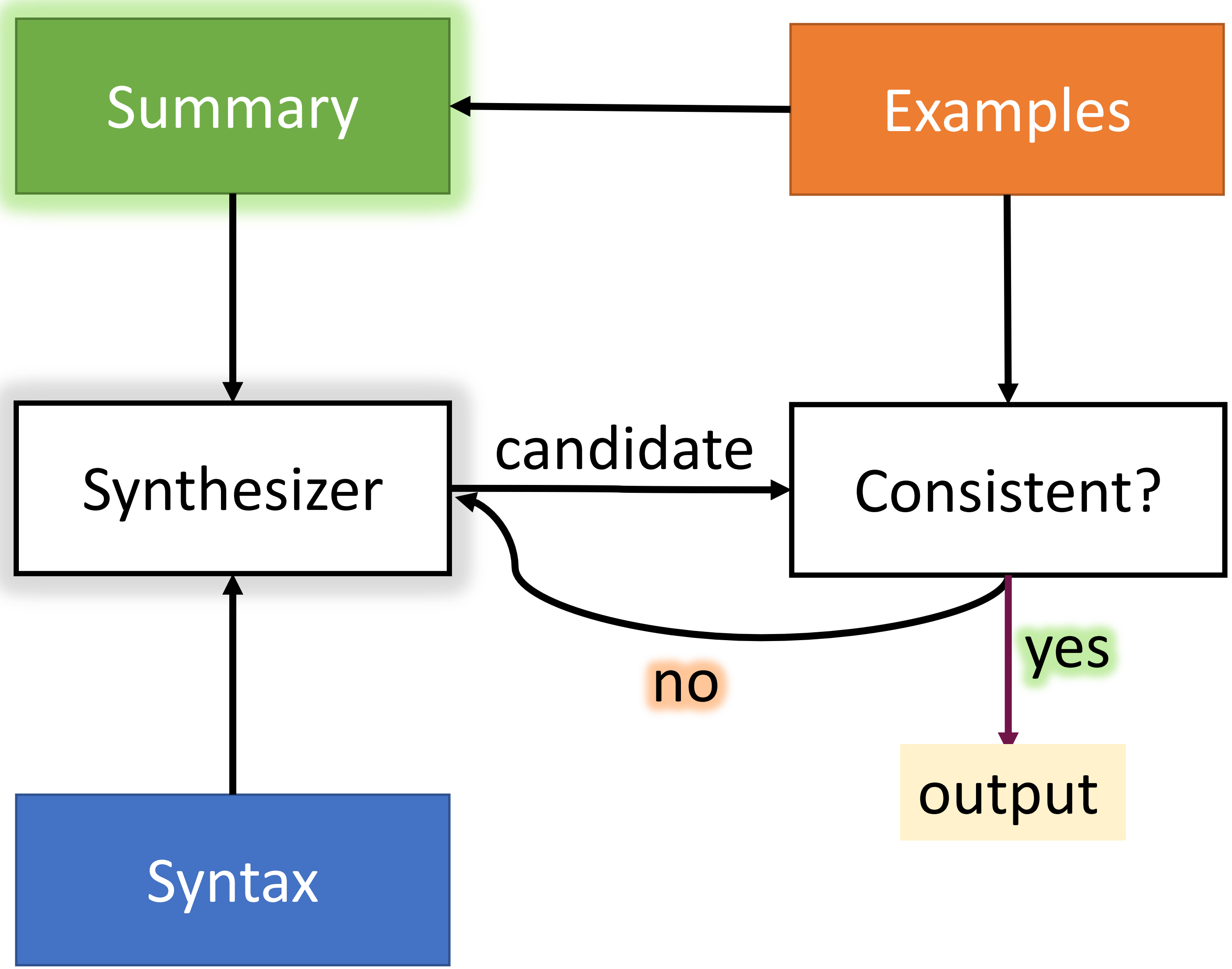
An Example

Crashes
Broadway
Whitehall St

Crashes(x) :− HasTraffic(x).

Crashes
Broadway
Wall St
William St
Whitehall St

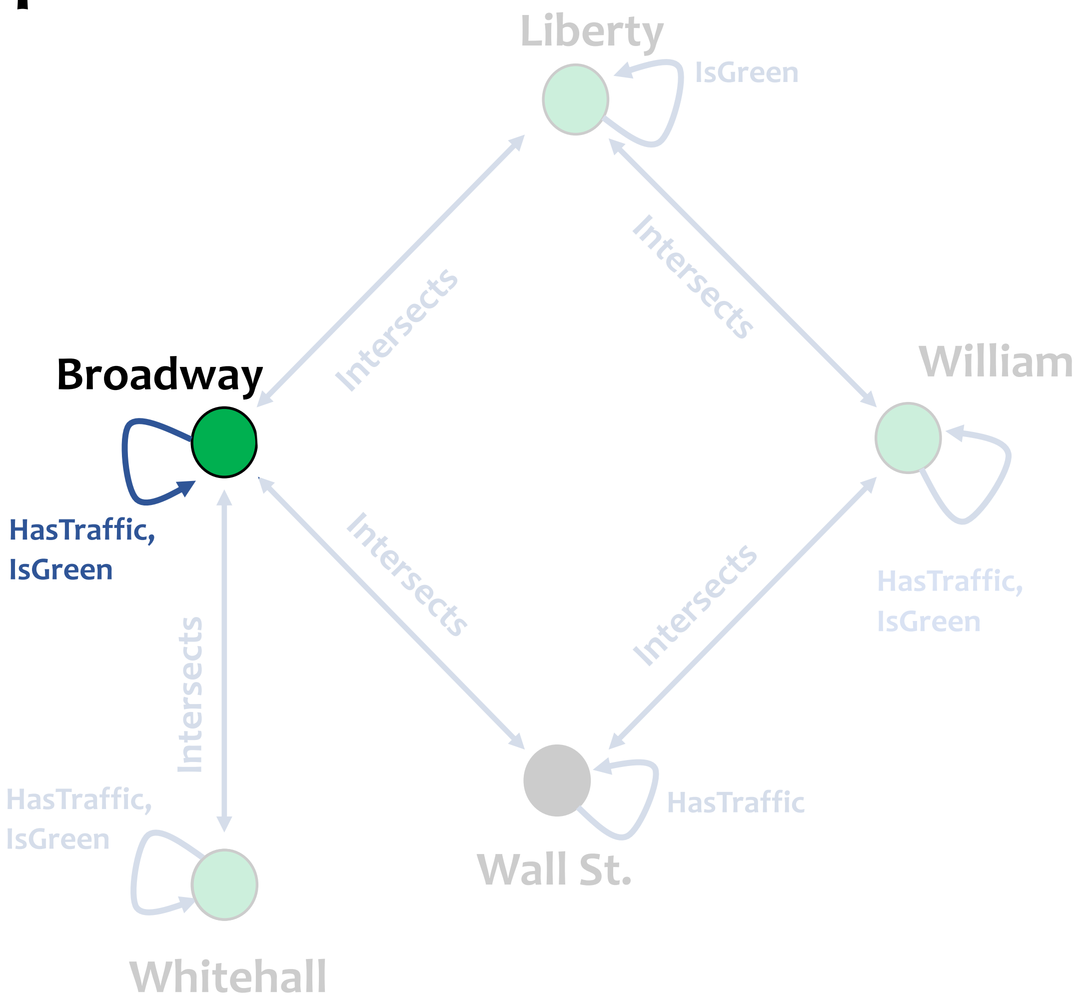



Summary
Examples
Synthesizer
candidate
Consistent?
yes
no
output
Syntax

An Example

Crashes
Broadway
Whitehall St

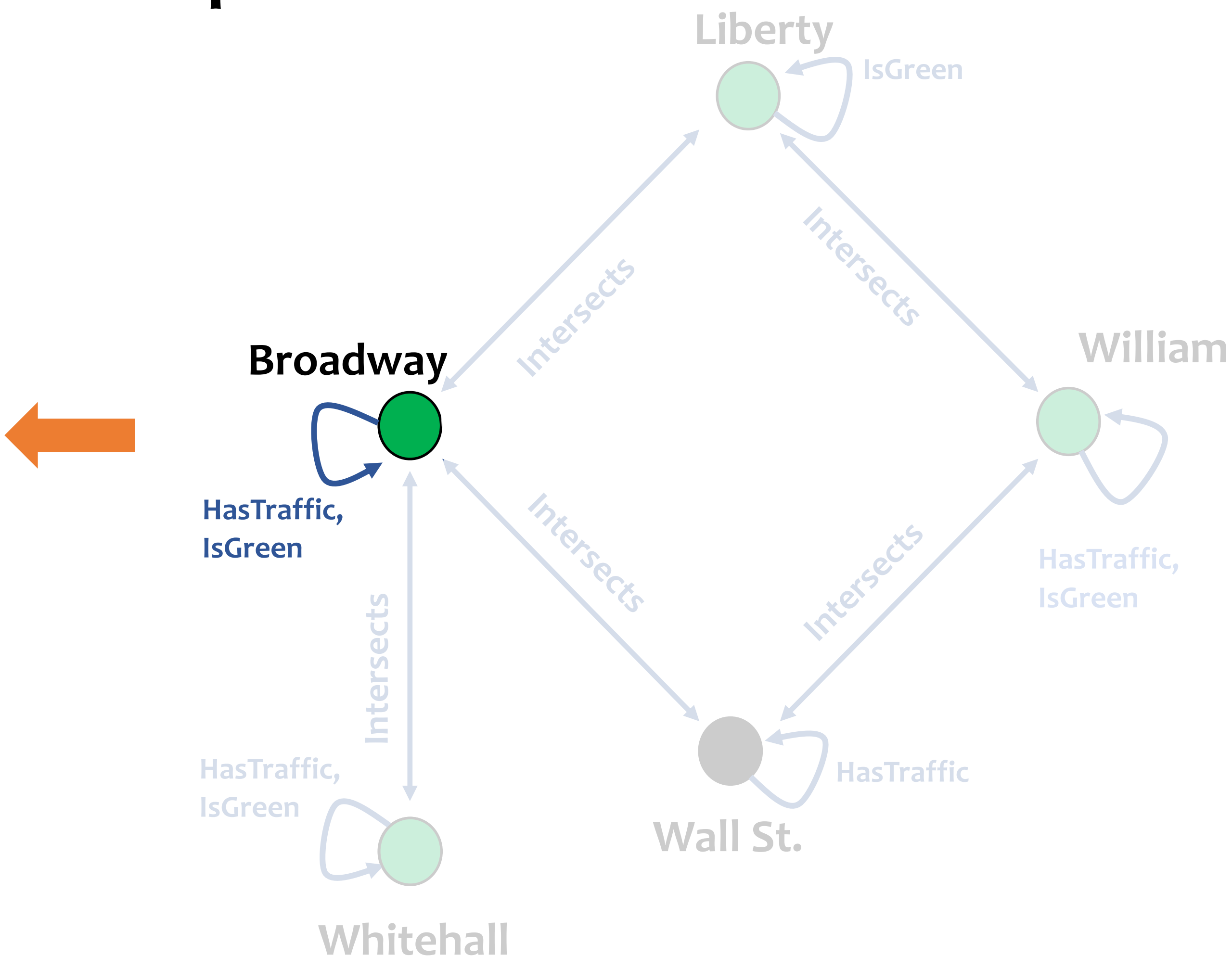
Crashes(x) :− HasTraffic(x), isGreen(x).



An Example

Crashes
Broadway
Whitehall St

Crashes(x) :– HasTraffic(x), isGreen(x).

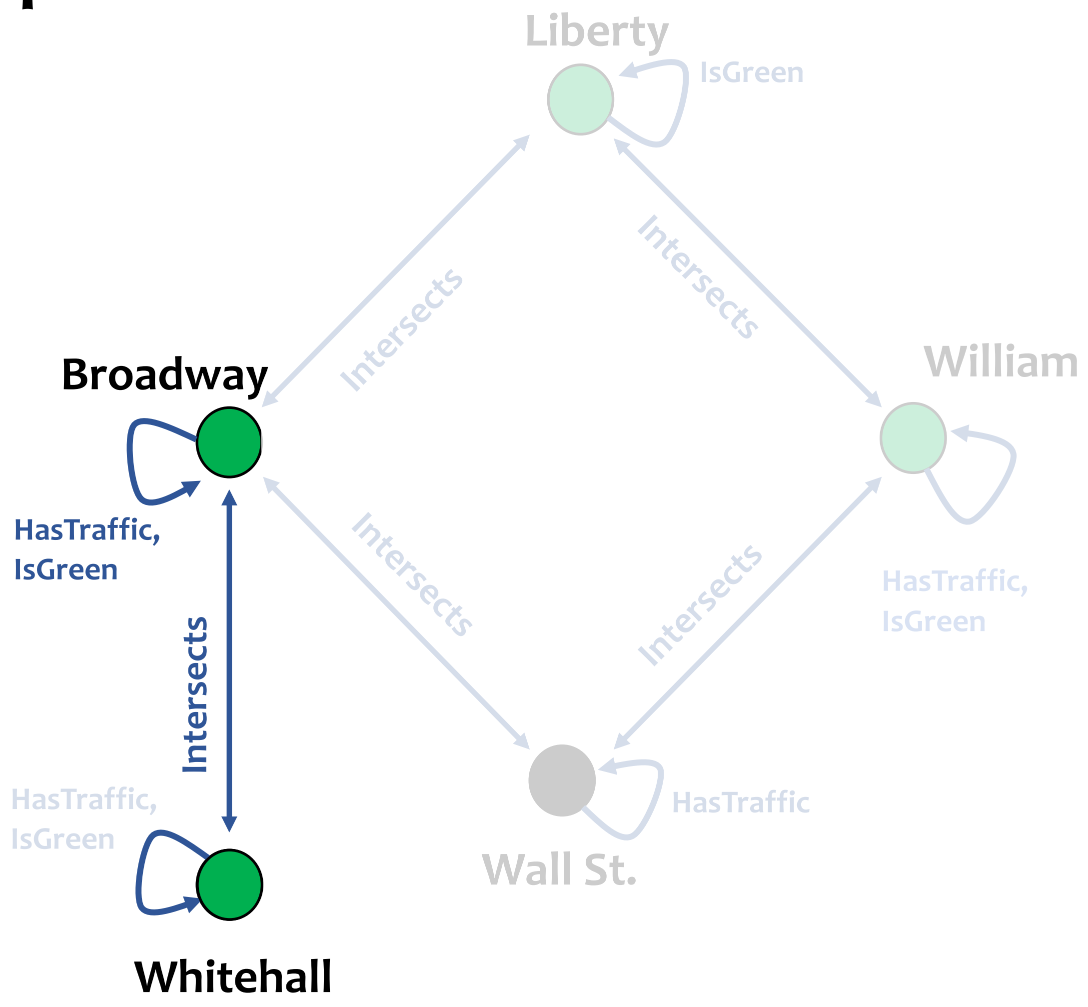


Crashes
Broadway
William St
Whitehall St

An Example

Crashes
Broadway
Whitehall St

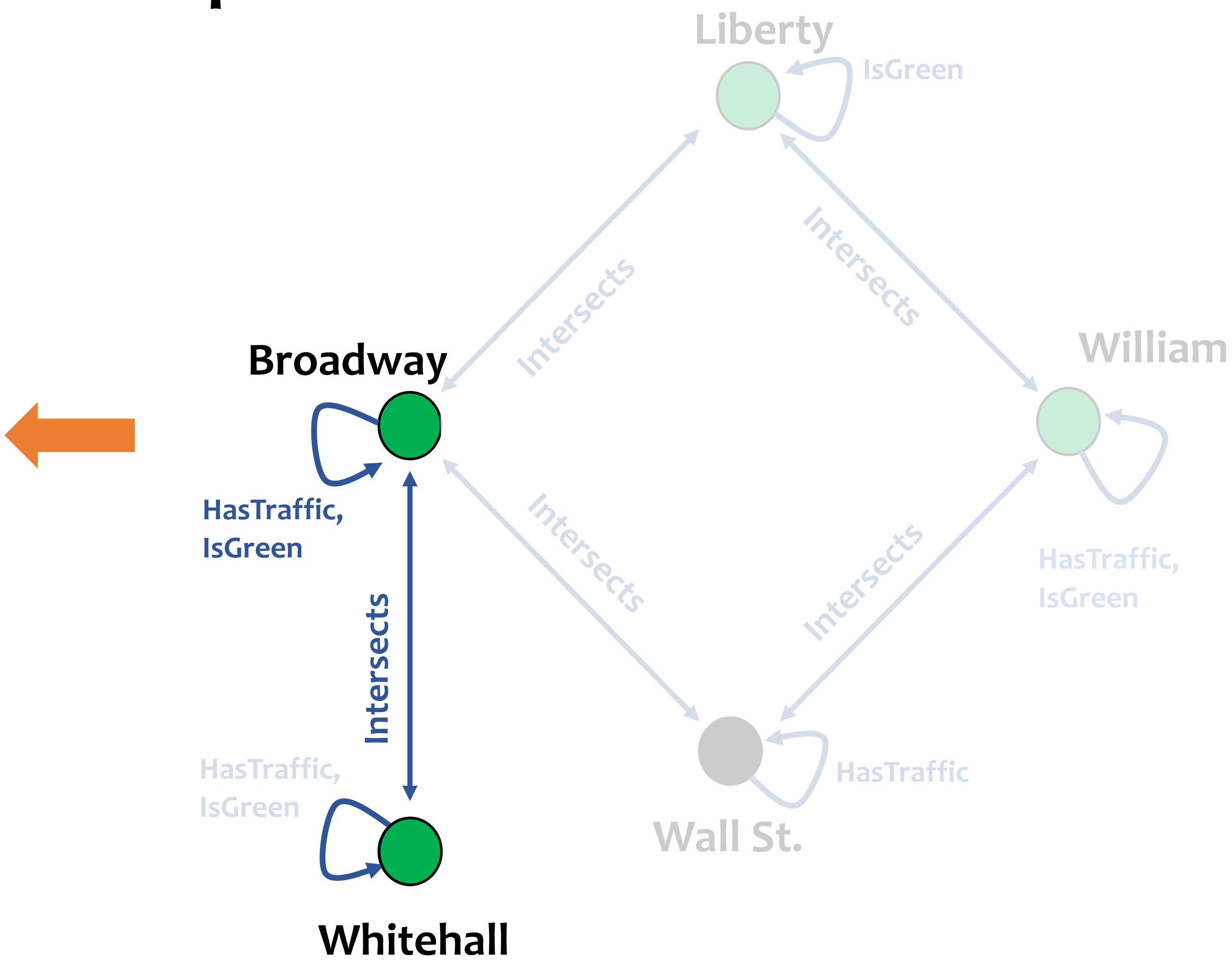
Crashes(x) :– HasTraffic(x), isGreen(x),
Intersects(x, y).



An Example

Crashes
Broadway
Whitehall St

Crashes(x) :− HasTraffic(x), isGreen(x), Intersects(x, y).

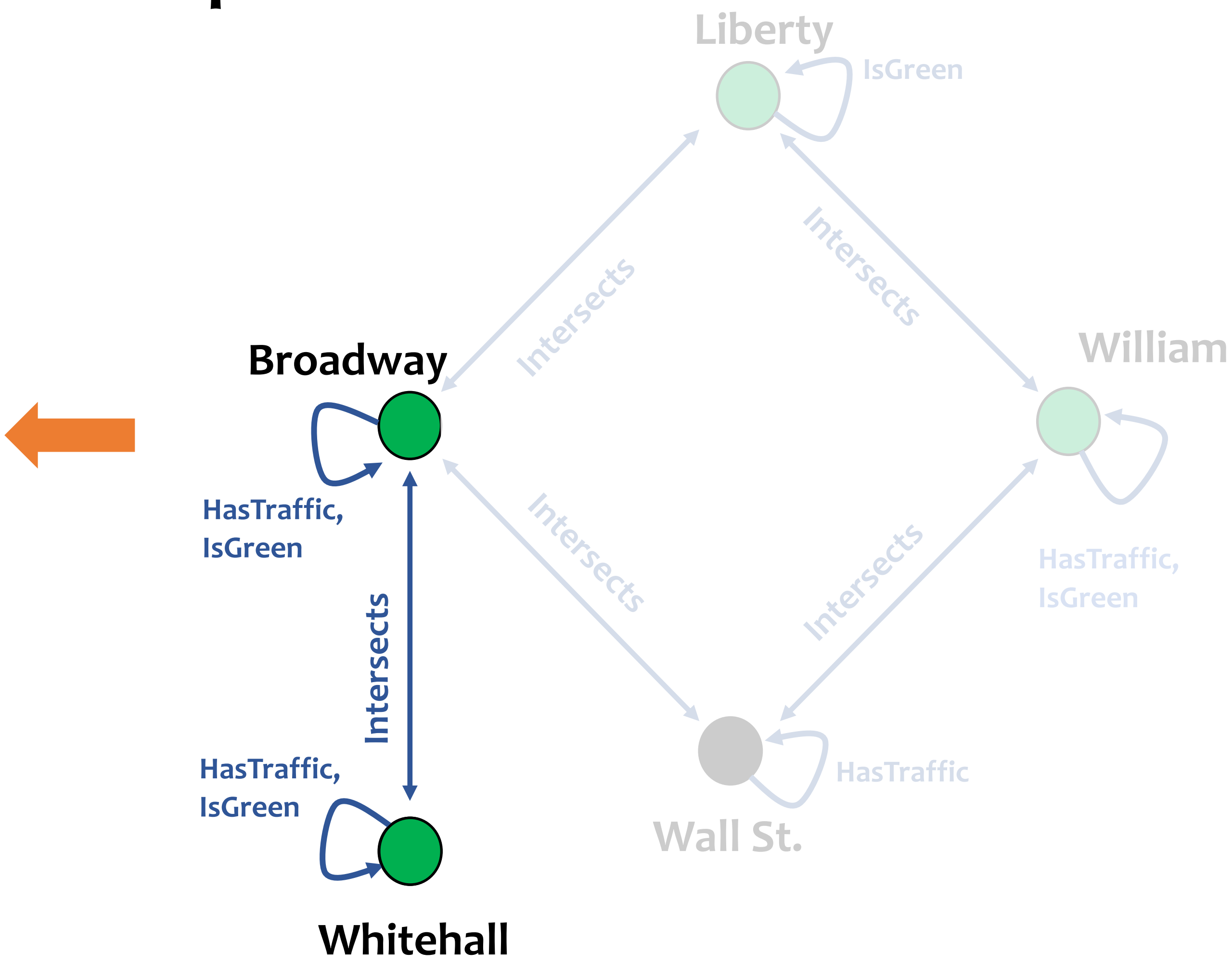


Crashes
Broadway
William St
Whitehall St

An Example

Crashes
Broadway
Whitehall St

Crashes(x) : – HasTraffic(x), isGreen(x),
Intersects(x, y),
HasTraffic(y), isGreen(y).

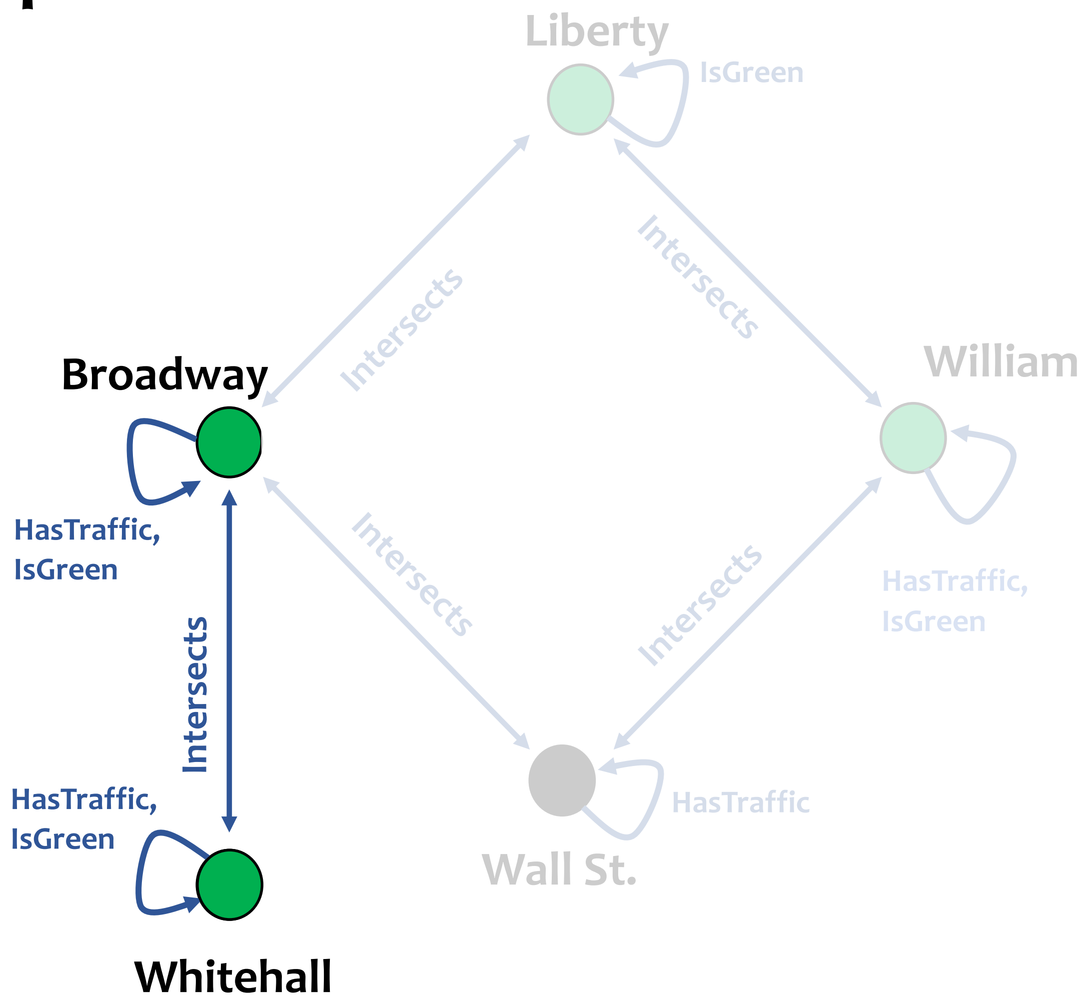


An Example

Crashes
Broadway
Whitehall St

Crashes(x) :− HasTraffic(x), isGreen(x),
Intersects(x, y),
HasTraffic(y), isGreen(y).

Crashes
Broadway
Whitehall St

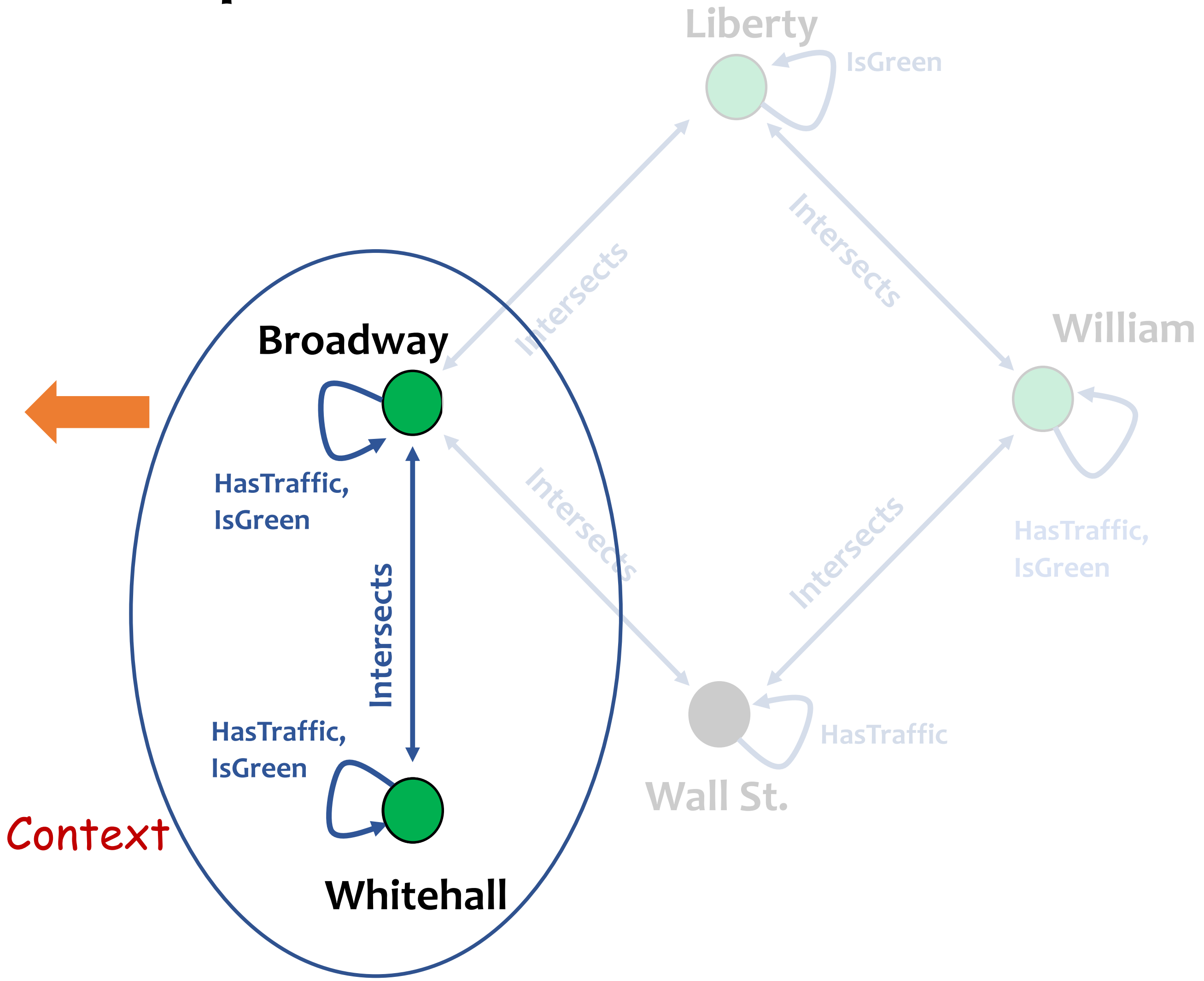


An Example

Crashes
Broadway
Whitehall St

Crashes(x) :– HasTraffic(x), isGreen(x),
Intersects(x, y),
HasTraffic(y), isGreen(y).

Crashes
Broadway
Whitehall St



Guarantees

1. EGS is **terminating** as there are finitely many subgraphs.

2. EGS is **sound** because consistency is verified as a part of synthesis.

3. EGS is **complete** because:

   the query corresponding to the entire graph is consistent with the examples if and only if some consistent query exists.

Complexity of Relational Query Synthesis

Aalok Thakkar, Rajeev Alur, Mayur Naik
University of Pennsylvania, Philadelphia, USA
{athakkar,alur,mhnaik}@cis.upenn.edu

1 INTRODUCTION

The synthesis of relational queries from input-output examples has been studied in the context of inductive logic programming [2, 4, 6–8], program synthesis [5, 9, 11–14], and neural learning [10]. It is a challenging problem, and analyzing the computational complexity of the problem for even restricted fragments can significantly impact the development of synthesis tools.

For instance, several tools use language biases such as mode declarations, templates, meta-rules, and candidate rules to constraint the space of candidate programs [4, 6, 7, 9, 11]. Studying the hardness of the problem can allow us to determine bounds on the language biases such that completeness of the search process is not compromised. Tools that search through an infinite space do not terminate if the instance of the synthesis problem is *unrealizable* [5, 14]. Studying the unrealizability of the problem instance can help determine when such a search is futile and should be abandoned, allowing us to give termination guarantees for these tools.

The above mentioned synthesis tools consider different frag-

predicate R with a list of k variables. Then, a rule r is of the form:

$$R_h(\vec{u}_h) \text{ :- } R_1(\vec{u}_1), R_2(\vec{u}_2), \ldots, R_n(\vec{u}_n),$$

where the single literal on the left, $R_h(\vec{u}_h)$, is the *head* of r and $R_1(\vec{u}_1)$, $R_2(\vec{u}_2)$, …, $R_n(\vec{u}_n)$, is called the *body* of r. The literals in the body can have input predicates, invented predicates, or output predicates, while the head of the rules must have either invented predicates or output predicates. A variable that occurs in the head must appear at least once in the body to bound the variables. The size of a rule is defined as the number of literals in its body. $|P|$, the size of a query P, is defined as the sum of the size of rules in its body.

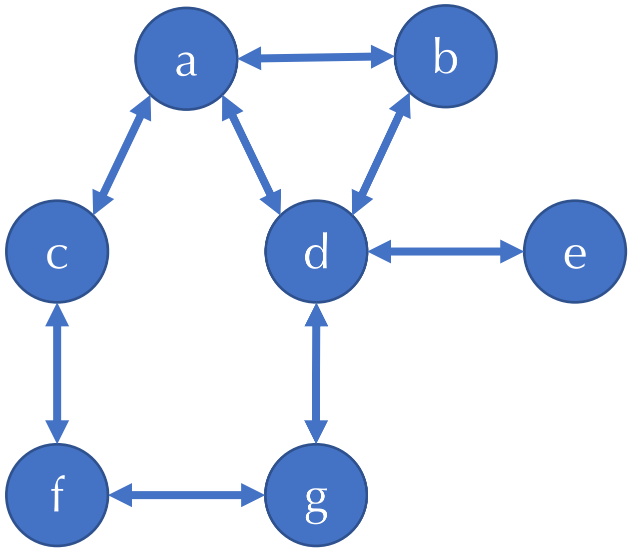
A fragment of these queries called *union of conjunctive queries* (UCQ) is of interest in Section 4. A UCQ consists of rules where the body of the rule comprises only of input predicates. UCQ is equivalent to select-project-join queries in relational algebra [3].

2.2 Semantics of Relational Queries

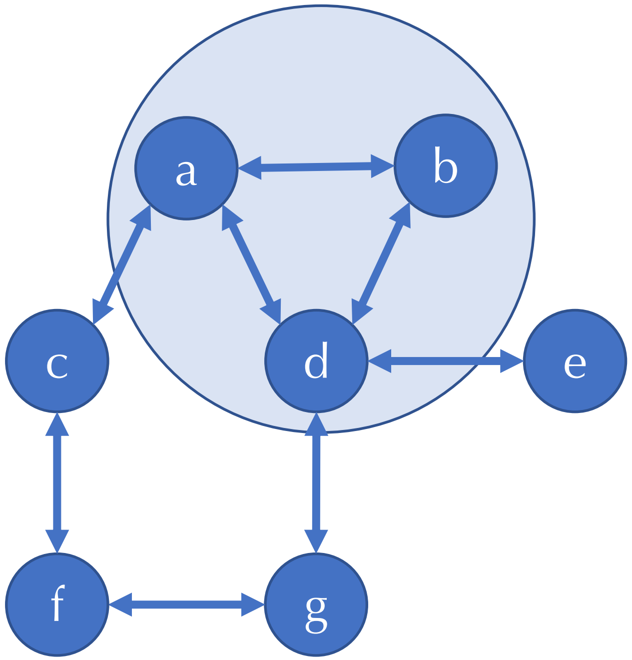
Decidability and Complexity

The instance of the synthesis problem is realizable if and only if the query corresponding to the **entire constant co-occurrence graph** is consistent with the input-output examples.

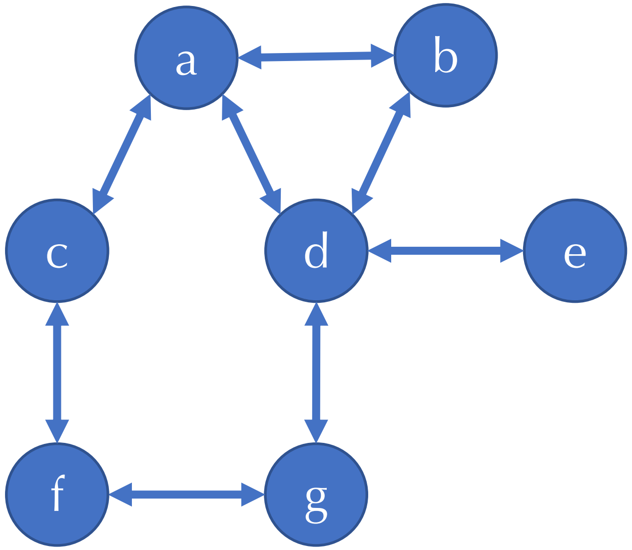
Hardness Result

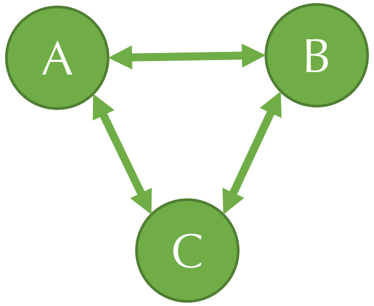


Hardness Result

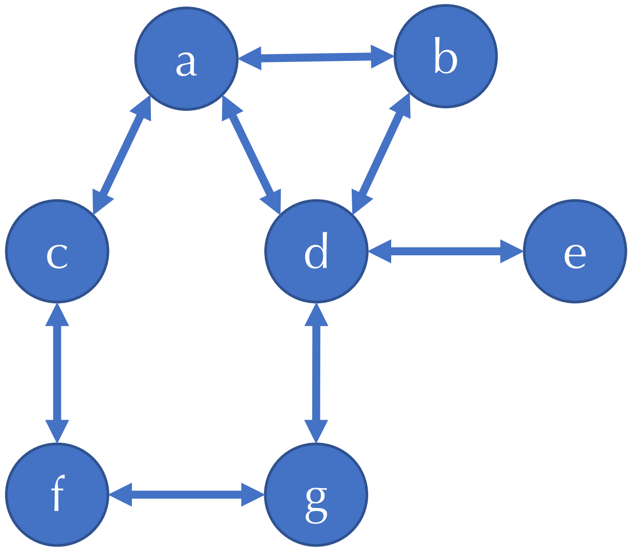


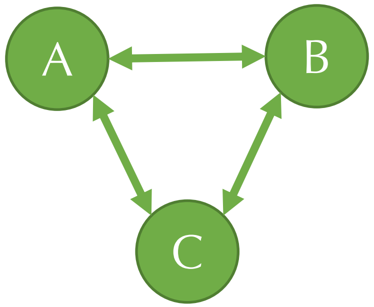
Hardness Result





Hardness Result





Target
A
B
C

Completeness Guarantees

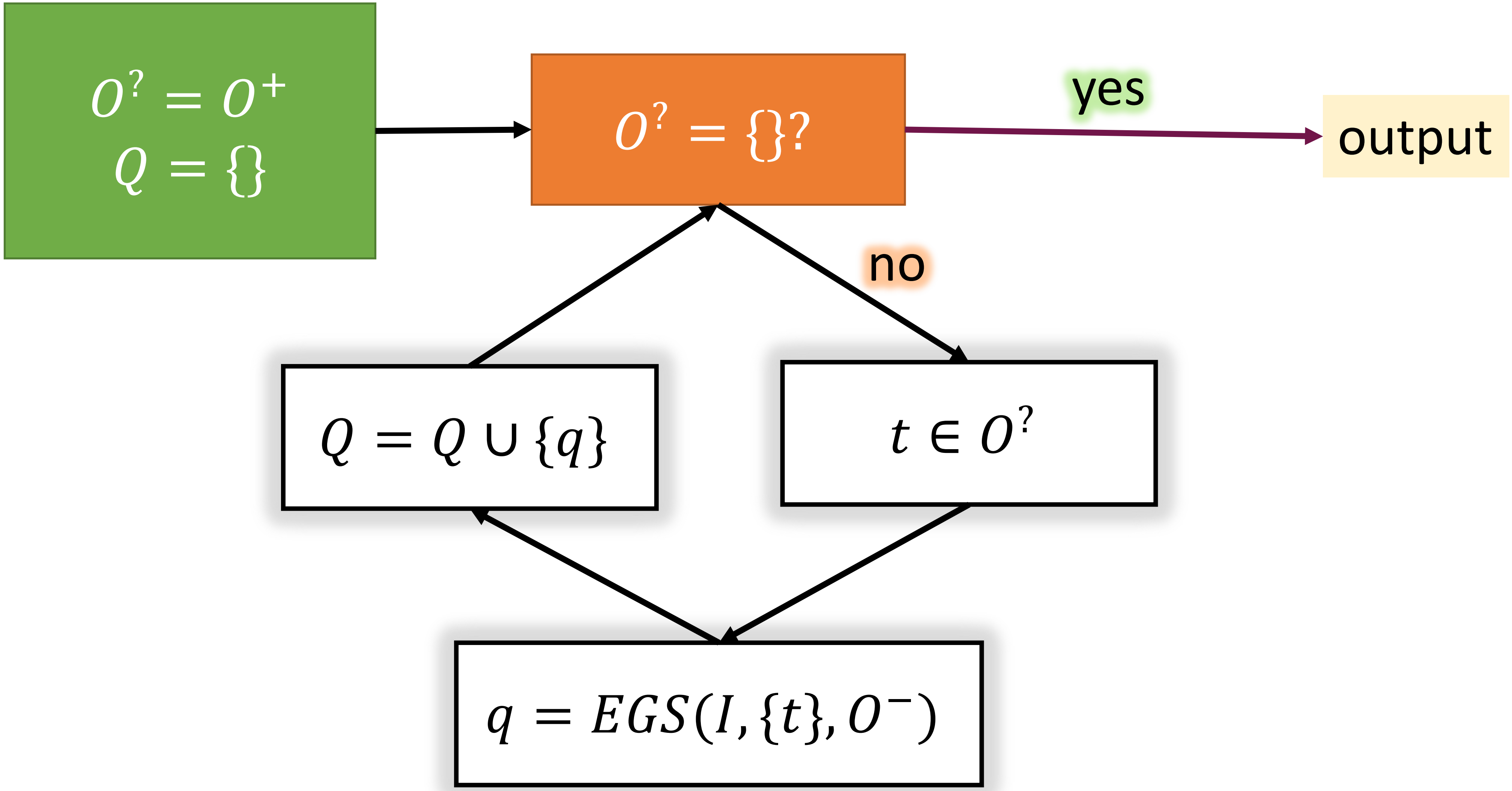
The instance of the synthesis problem is realizable if and only if there exists a query of size at most $|I \times O^+|$ that is consistent with the examples.

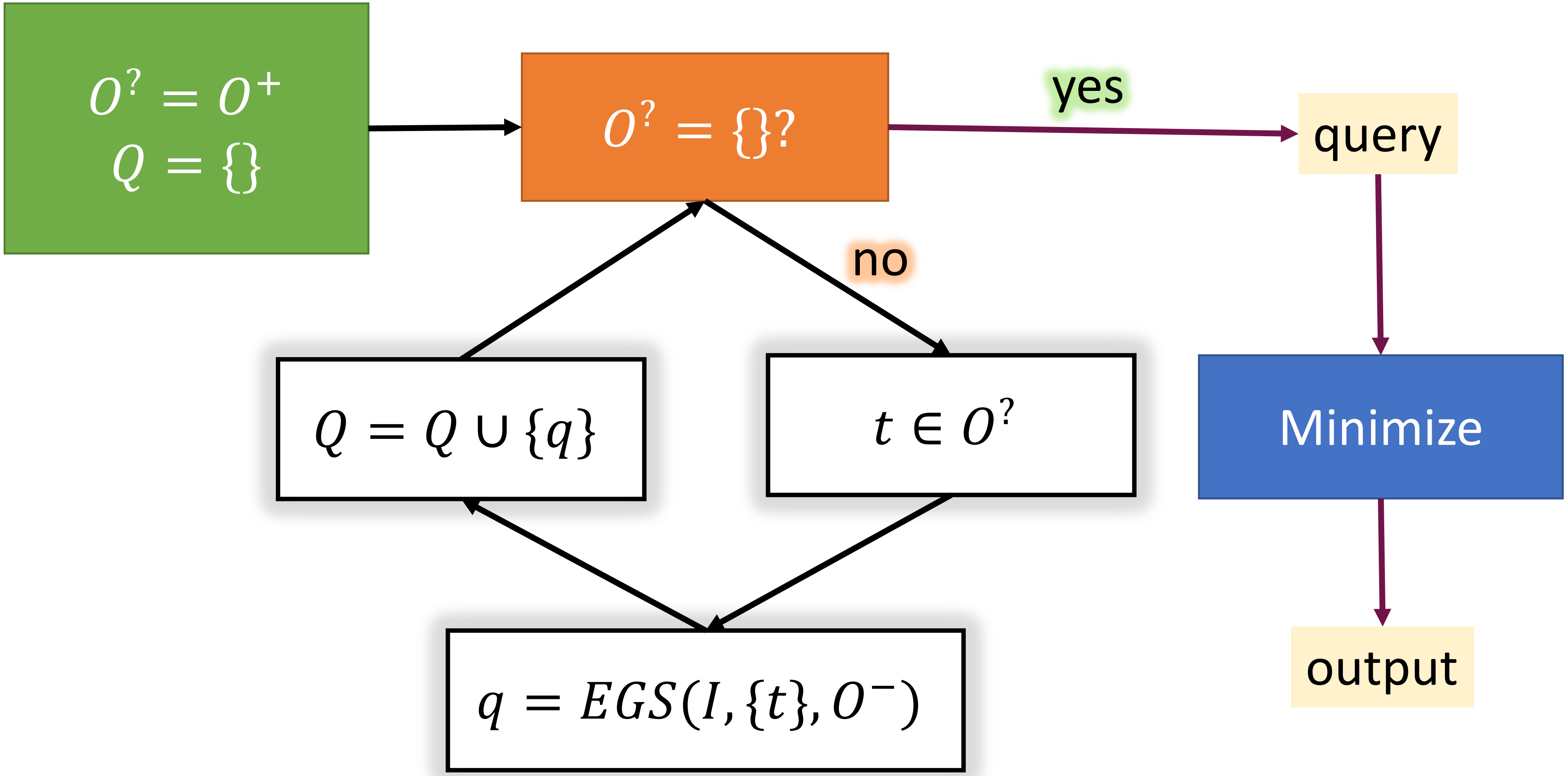
Extensions

Union

Recursion

Comparison Predicates


$O^? = O^+$
$Q = \{\}$
$O^? = \{\}?$
yes
output
no
$Q = Q \cup \{q\}$
$t \in O^?$
$q = EGS(I, \{t\}, O^-)$



$O^? = O^+$
$Q = \{\}$
$O^? = \{\}?$
yes
no
query
$Q = Q \cup \{q\}$
$t \in O^?$
Minimize
$q = EGS(I, \{t\}, O^-)$
output

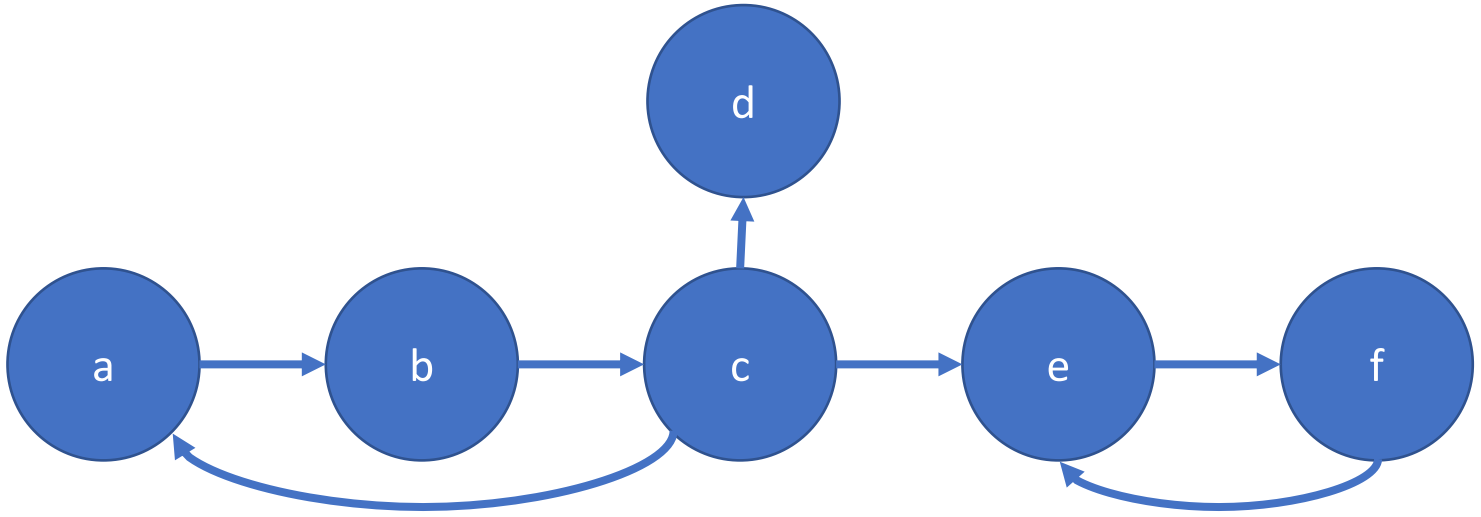
Extensions

Union

Recursion

Comparison Predicates

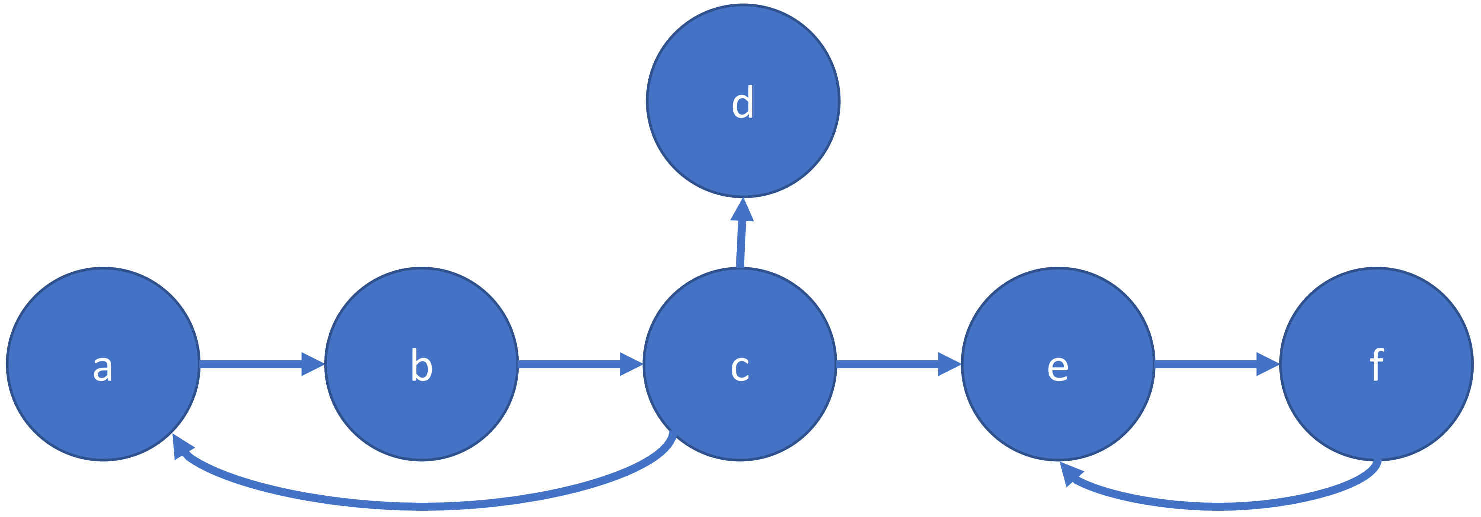
```
scc(x, y) :– path(x, y), path(y, x).
path(x, y) :– edge(x, y).
path(x, y) :– egde(x, y), egde(y, z).
```



scc(x, y) :– path(x, y), path(y, x).

path(x, y) :– edge(x, y).

path(x, y) :– path(x, z), path(z,y).



scc(x, y) :– edge(x, y), edge(y, x).

scc(x, y) :– edge(x, y), edge(y, z), edge(z, x).

scc(x, z) :– edge(x, y), edge(y, z), edge(z, x).

Normalization

scc(x, y) : − edge(x, y), edge(y, x).
scc(x, y) : − edge(x, y), edge(y, z), edge(z, x).
scc(x, z) : − edge(x, y), edge(y, z), edge(z, x).

Normalization

$$\begin{aligned}
scc(x, y) &:- R(x, y), R(y, x). \\
scc(x, y) &:- R(x, y), R(y, z), R(z, x). \\
scc(x, z) &:- R(x, y), R(y, z), R(z, x). \\
R(x, y) &:- edge(x, y).
\end{aligned}$$

Normalization

scc(x, y) :− R(x, y), R(y, x).
scc(x, y) :− R(x, y), S(y, x).
scc(x, z) :− S(x, z), R(z, x).
R(x, y) :− edge(x, y).
S(x, z) :− R(x, y), R(y, z).

Normalization

$$scc(x, y) \ :- \ R(x, y), R(y, x).$$
$$scc(x, y) \ :- \ S(x, z), R(z, x).$$
$$scc(x, z) \ :- \ S(x, z), R(z, x).$$
$$R(x, y) \ :- \ edge(x, y).$$
$$S(x, z) \ :- \ R(x, y), R(y, z).$$

Unification

```
scc(x, y)  :− P(x, y), P(y, x).
scc(x, y)  :− P(x, y), P(z, y).
scc(x, z)  :− P(x, z), P(z, x).
P(x, y)    :− edge(x, y).
P(x, z)    :− P(x, y), P(y, z).
```

Unification

```
scc(x, y)  : − P(x, y), P(y, x).
P(x, y)    : − edge(x, y).
P(x, z)    : − P(x, y), P(y, z).
```

Extensions

Union

Recursion

Comparison Predicates

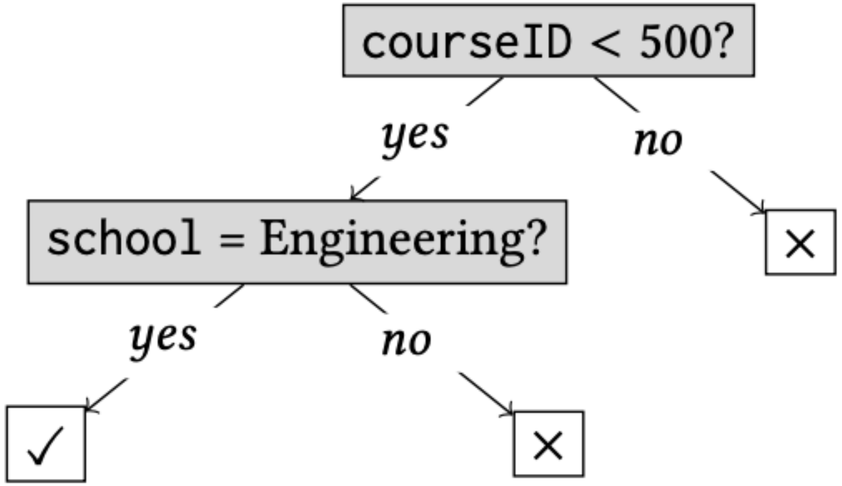
```sql
SELECT registration.studentID
    FROM registration JOIN department
        ON registration.deptCode = department.deptCode
    WHERE registration.courseID < 500
        AND department.school = "Engineering"
```

```sql
SELECT registration.studentID
    FROM registration JOIN department
        ON registration.deptCode = department.deptCode
```

```
SELECT registration.studentID
    FROM registration JOIN department
        ON registration.deptCode = department.deptCode
    WHERE registration.courseID < 500
        AND department.school = "Engineering"
```

```
SELECT registration.studentID
    FROM registration JOIN department
        ON registration.deptCode = department.deptCode
    WHERE registration.courseID < 500
        AND department.school = "Engineering"
```

studentID	deptCode	courseID	school
Alice	Comp.	201	Engineering
Alice	Chem.	310	Arts and Science
Alice	Mech.	550	Engineering
Bob	Mech.	320	Engineering
Bob	Mech.	550	Engineering
Charlie	Chem.	310	Arts and Science
David	Comp.	500	Engineering
David	Mech.	502	Engineering
Erin	Chem.	310	Arts and Science



Future Work

Comprehensive Evaluation

Application

Aggregation