From Operational to Declarative Specifications using a Genetic Algorithm
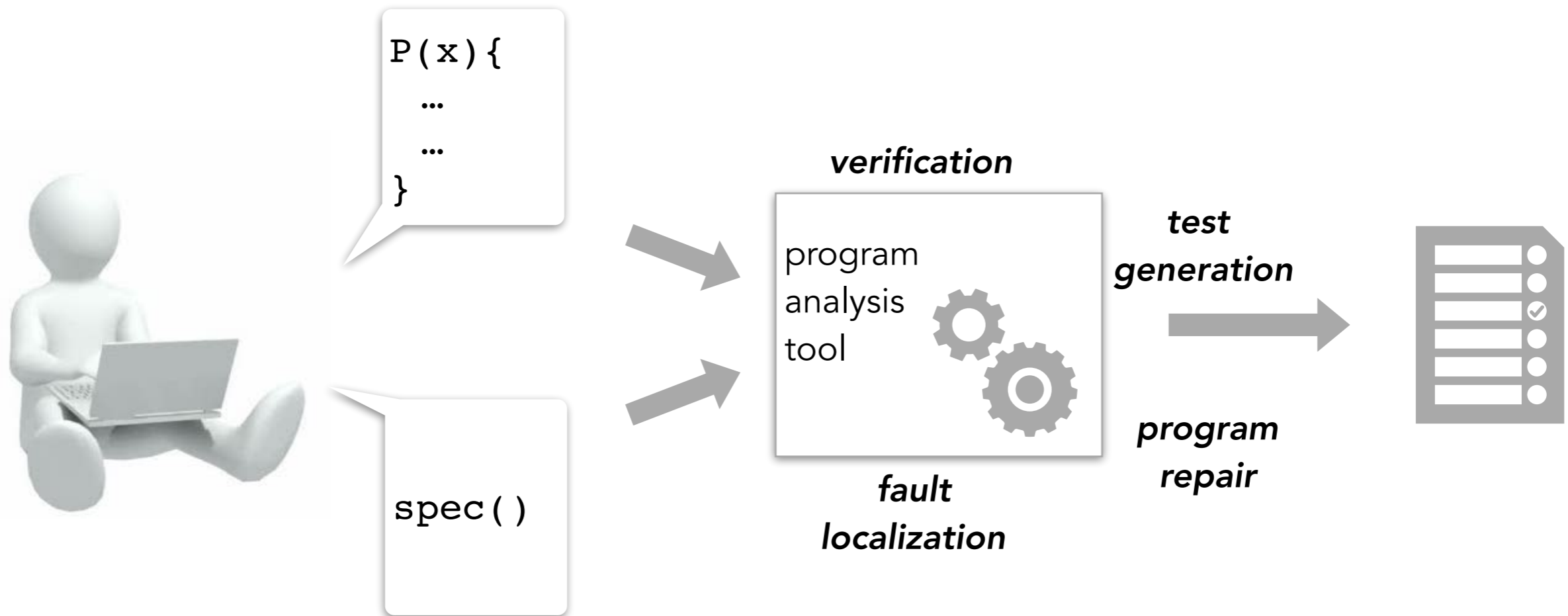
Renzo Degiovanni

June, 2018

# Motivation

Program Analysis


```
P(x){
  …
  …
}
```

spec()

verification

program
analysis
tool

test
generation

program
repair

fault
localization

Specification formalisms

Operational

```java
public boolean repOK () {
    Set<Entry> visited = new HashSet<Entry>();
    visited.add(header);
    Entry current = header;
    while (true) {
        Entry next = current.getNext();
        if (next == null) break;
        if (!visited.add(next)) return false;
        current = next;
    }
    if (visited.size() != size) return false;
    return true;
}
```
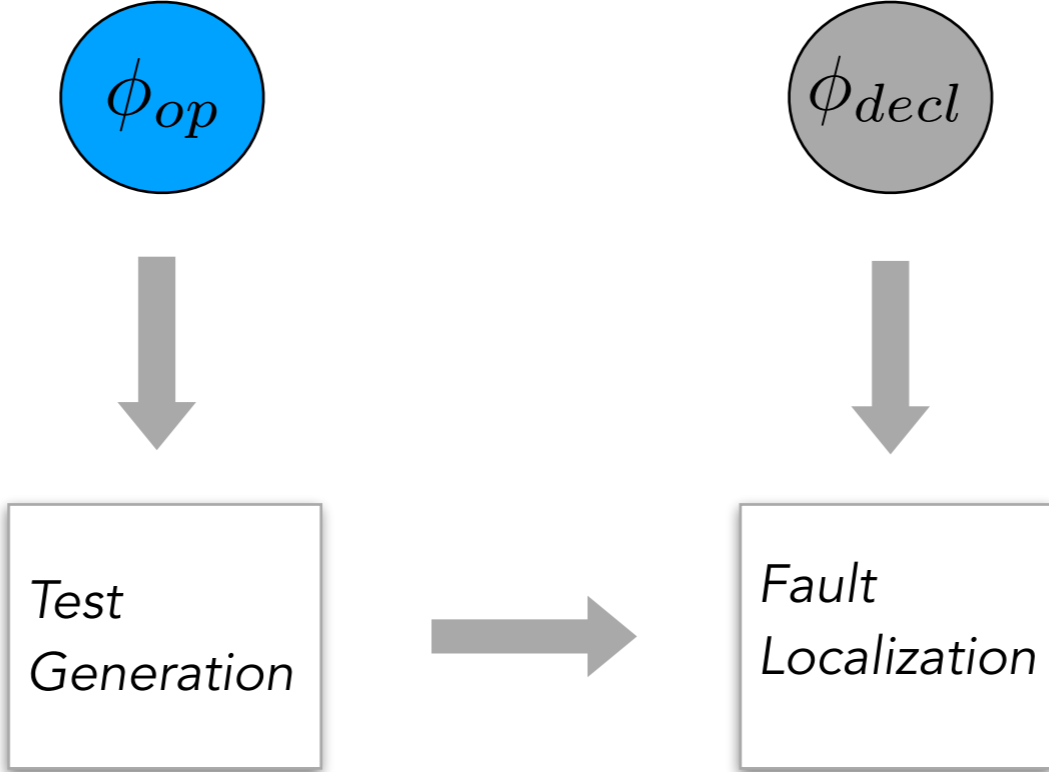
Korat
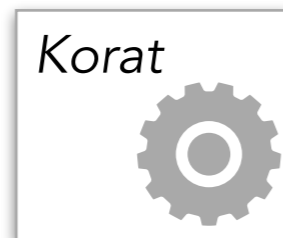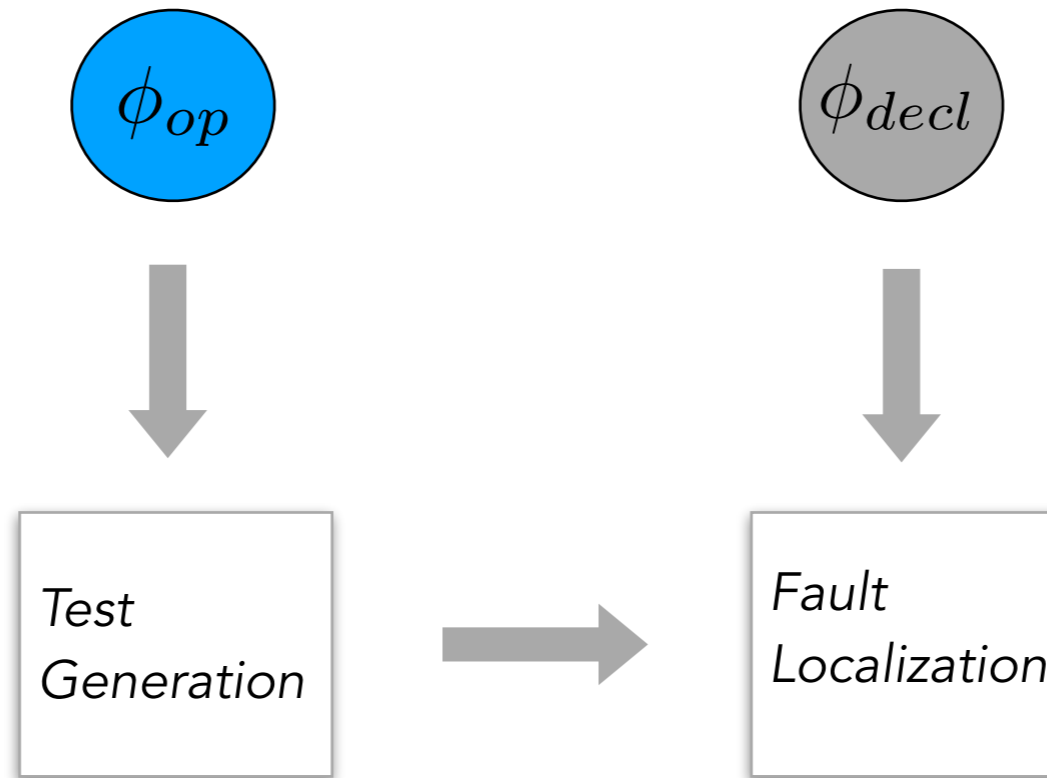
Declarative

```
pred repOK [
    thiz:List,
    header: List -> one Node+Null,
    size: List -> one Int,
    next: Node -> one Node + Null]{
    (all n: thiz.header.*next | n !in n.^next)
        and
    (# thiz.header.*next = thiz.size)
}
```
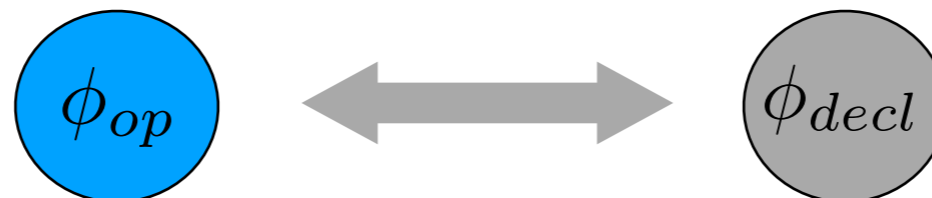
TestEra

# Cross-usage of automated analysis tools

Cross-usage of automated analysis tools

ϕ_{op}

ϕ_{decl}

Test Generation → Fault Localization

• What if we have the specification in just one style?
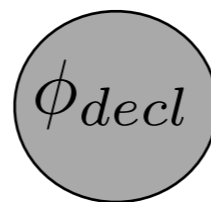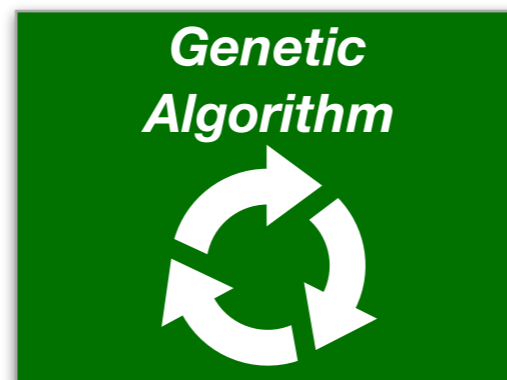
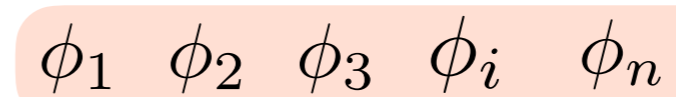ϕ_{op} ⟷ ϕ_{decl}

Approximating an Operational Specification by Evolving Declarative Specifications

Operational specification

ϕ_{op}

Catalog properties

$\phi_1 \quad \phi_2 \quad \phi_3 \quad \phi_i \quad \phi_n$

Genetic Algorithm

ϕ_c

ϕ_{decl}
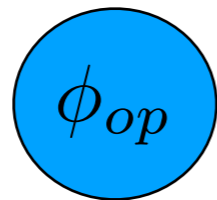
Declarative specification
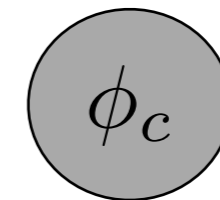
Structure of our Genetic Algorithm

Catalog properties

$$\phi_1 \ \phi_2 \ \phi_3 \ \phi_4$$

Search space

$$\phi_1 \wedge \phi_2$$

$$\phi_4$$

$$\neg\phi_3$$

$$\neg\phi_1 \wedge \neg\phi_4$$

$$\phi_2 \wedge \neg\phi_4$$

Structure of our Genetic Algorithm

Catalog properties

$$\phi_1 \ \phi_2 \ \phi_3 \ \phi_4$$

Search space

$$\phi_1 \wedge \phi_2$$

$$\phi_4$$

$$\neg\phi_3$$

$$\neg\phi_1 \wedge \neg\phi_4$$

$$\phi_2 \wedge \neg\phi_4$$

How we represent candidate specifications as chromosomes?

- **Chromosomes** are vectors of {T, F, *} and its size is equal to the number of properties in the catalog

$$\phi_1 \; \phi_2 \; \phi_3 \; \phi_4$$

| T | * | F | T |

- **Chromosomes** are vectors of {T, F, *} and its size is equal to the number of properties in the catalog

$$\phi_1 \; \phi_2 \; \phi_3 \; \phi_4 \qquad\qquad \boxed{T \; | \; * \; | \; F \; | \; T}$$

- The **specification** represented by the chromosome is a conjunction involving the catalog properties

$$\phi_1 \quad \phi_2 \quad \phi_3 \quad \phi_4$$

$$\boxed{T \; | \; * \; | \; F \; | \; T}$$

$$\downarrow$$

$$\phi_1 \wedge \neg\phi_3 \wedge \phi_4$$

Structure of our Genetic Algorithm

Catalog properties

$\phi_1 \ \phi_2 \ \phi_3 \ \phi_4$

Search space

$\phi_1 \wedge \phi_2$

ϕ_4

$\neg \phi_3$

$\neg \phi_1 \wedge \neg \phi_4$

$\phi_2 \wedge \neg \phi_4$

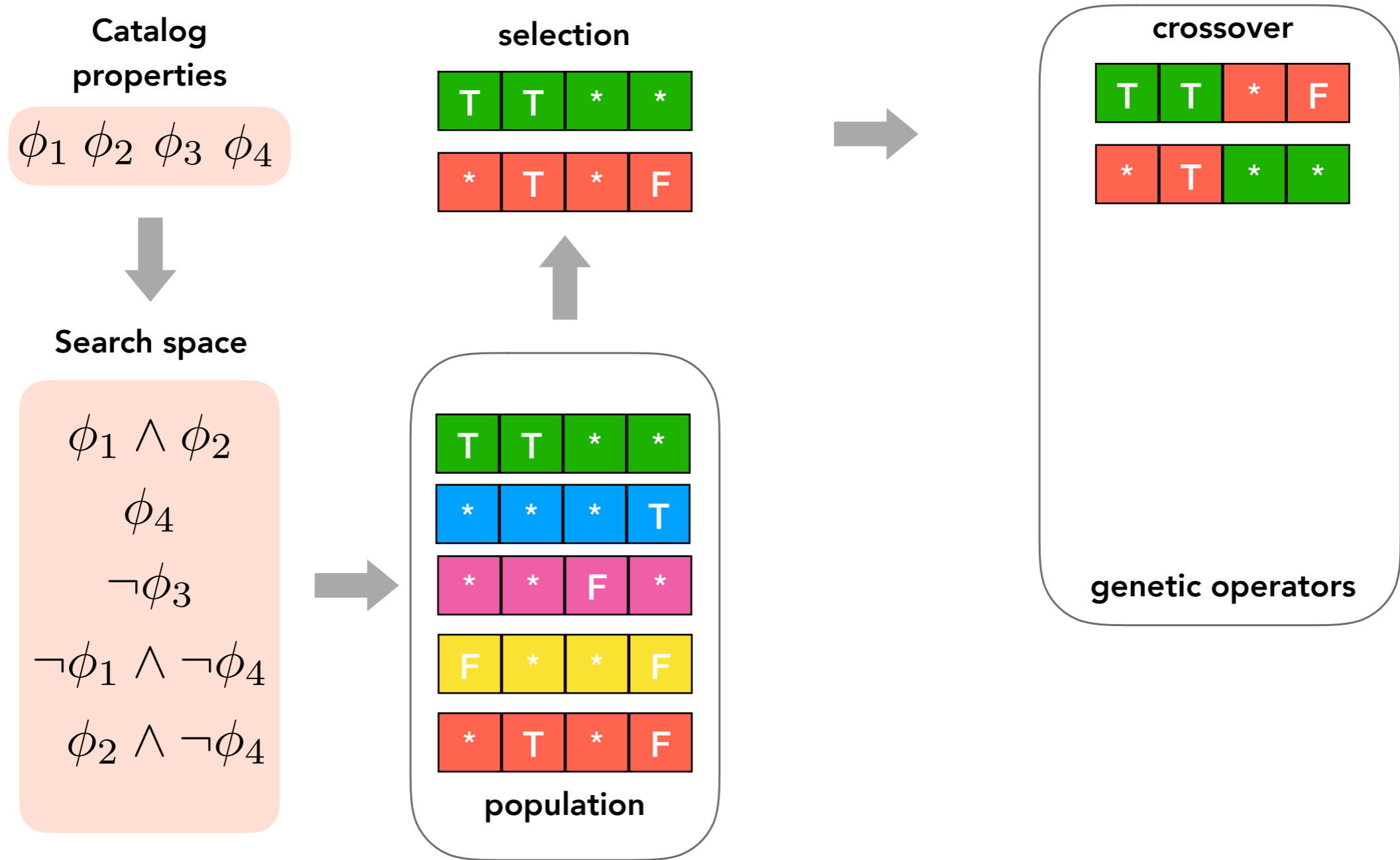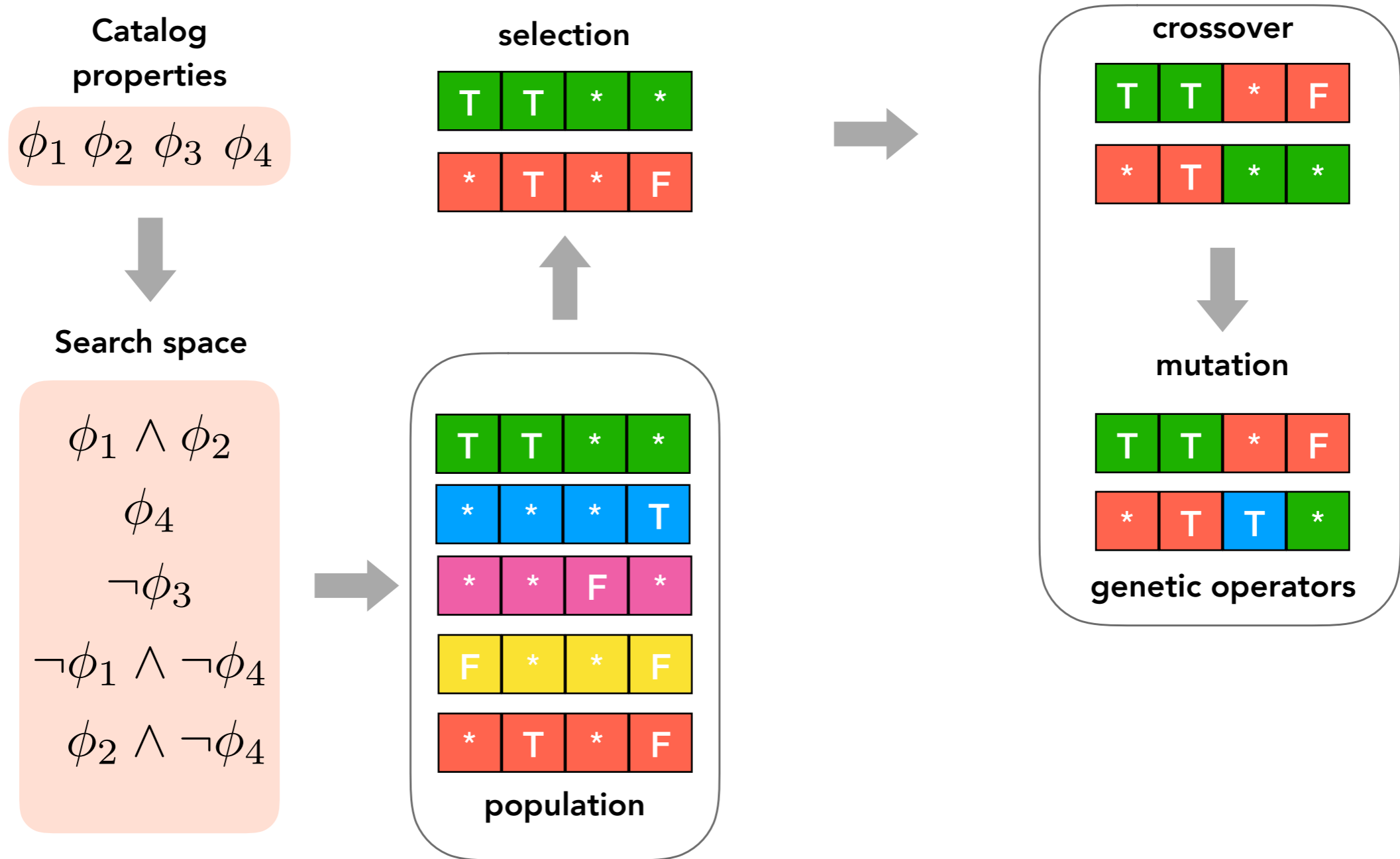| T	T	*	*
*	*	*	T
*	*	F	*
F	*	*	F
*	T	*	F

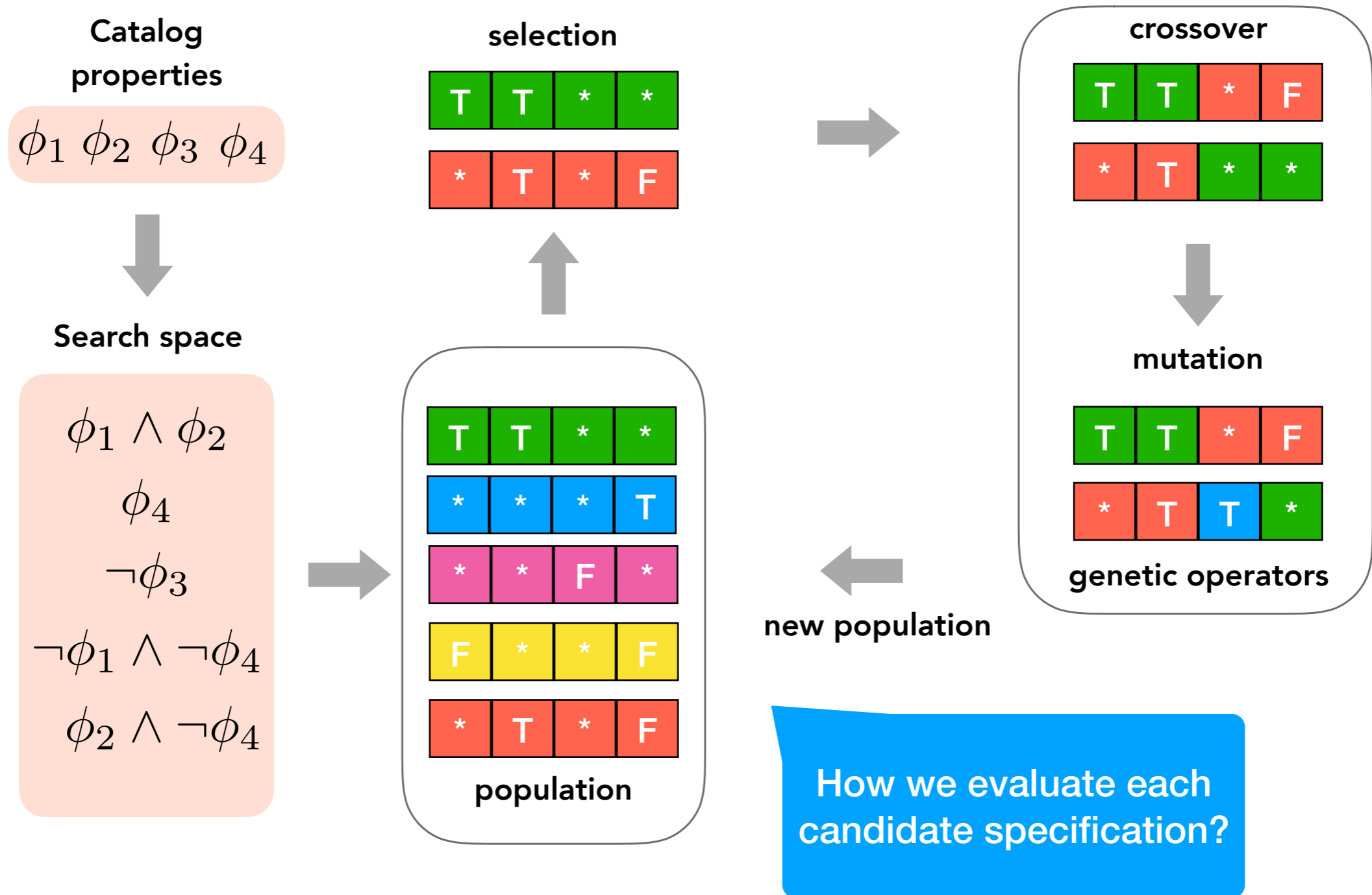population

Structure of our Genetic Algorithm
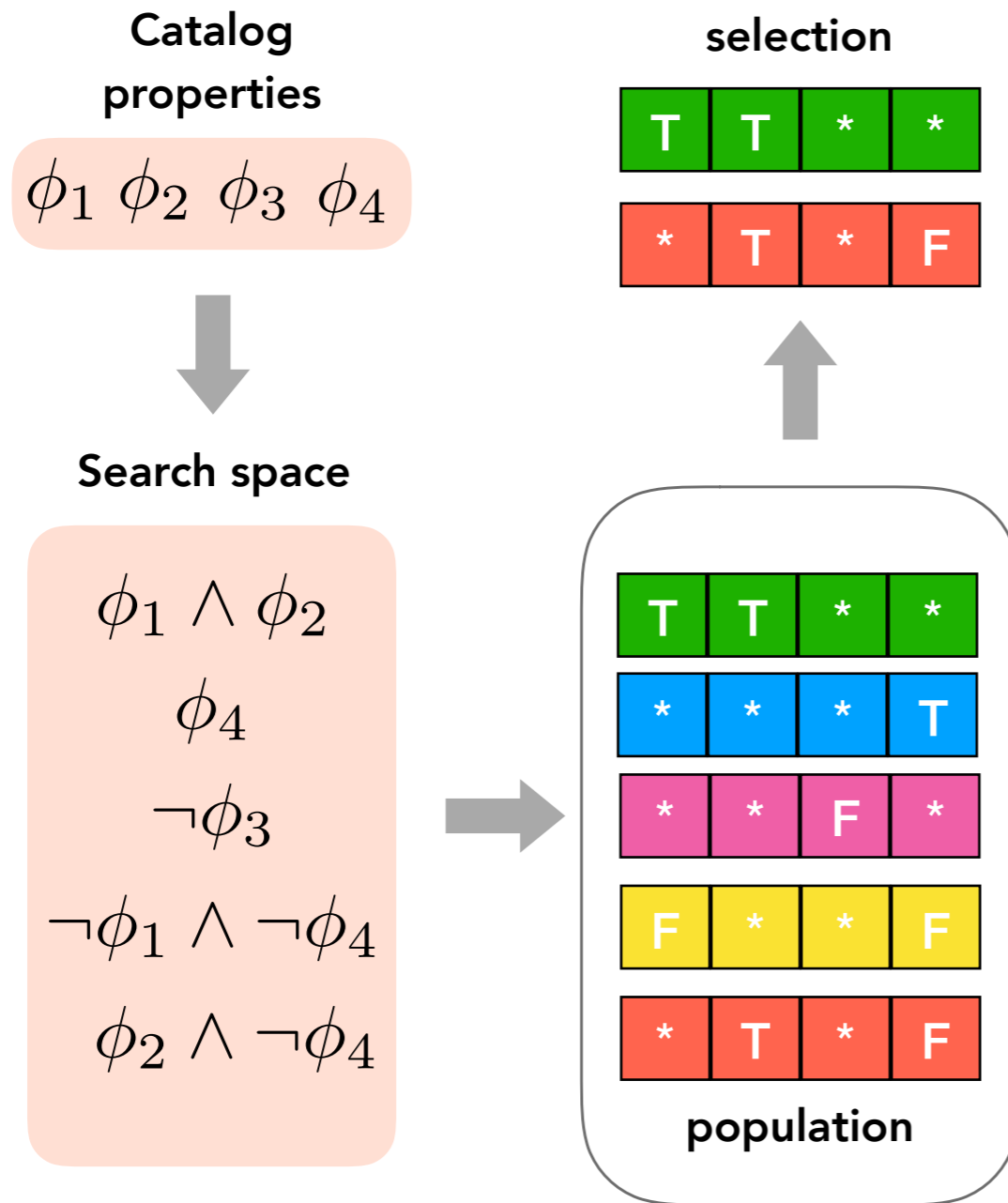
Structure of our Genetic Algorithm

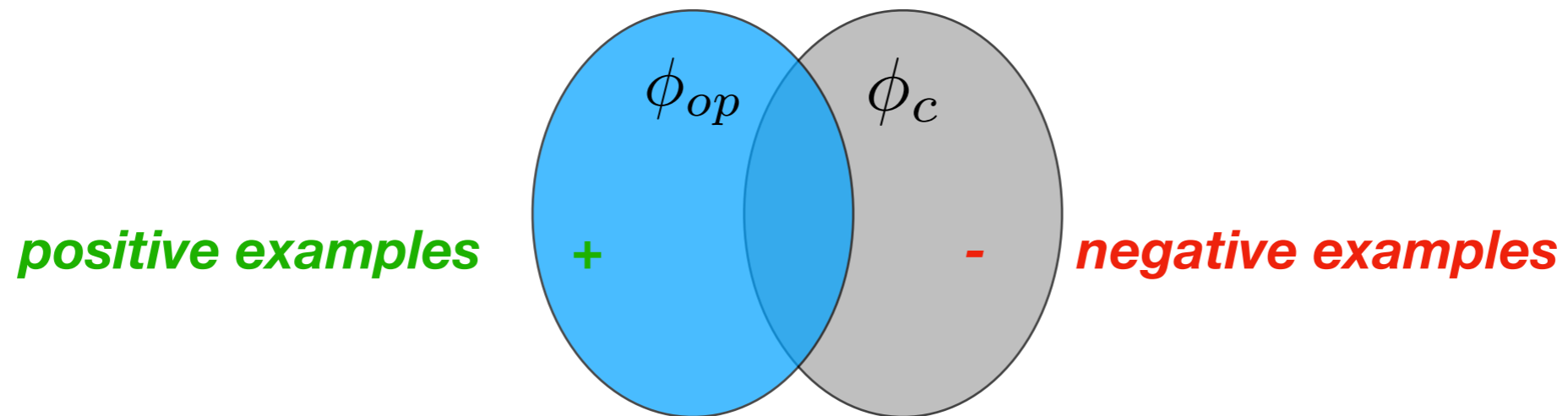# Structure of our Genetic Algorithm

Structure of our Genetic Algorithm

Fitness of Candidate Specifications

- For a chromosome *c* representing the specification *s*, assess how close is *s* to the desired specification

ϕ_{op} ϕ_c

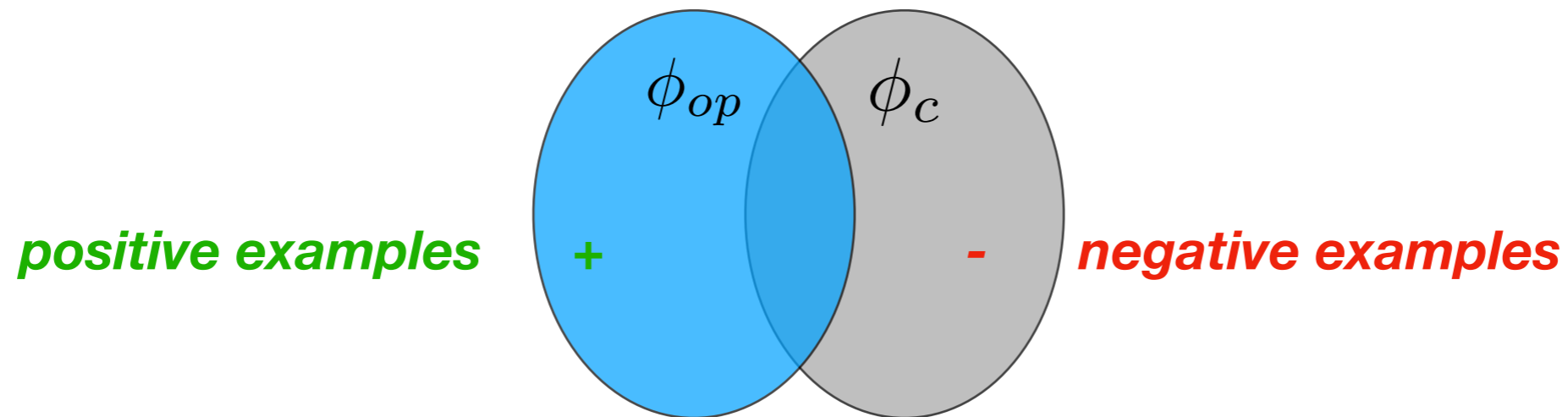*positive examples*   +   -   *negative examples*

Fitness of Candidate Specifications

- For a chromosome *c* representing the specification *s*, assess how close is *s* to the desired specification

positive examples $\qquad$ + $\qquad$ - $\qquad$ negative examples

ϕ_{op} $\qquad$ ϕ_c

- The fitness value of *c* is computed by counting the positive and negatives examples:

    ✔ The fewer the examples, the better

    ✔ The shorter the specification, the better

Evaluation

- Focused on invariants of data structures of varying complexity starting from a catalog with *common* invariant properties

		Invariant found?	23 catalog properties
linear structures	*singly linked lists*	✓	*(a)cyclicity*
	sorted linked lists	✓	*circularity*
	circular linked lists	✓	*size related props.* *ordering*

			28 catalog properties
n-ary (tree-like) structures	*binary trees*	✓	*(a)cyclicity*
	binary search trees	✓	*circularity*
	heaps	✓	*size related props.*
	binary dags	✓	*ordering* *disjointness*
	red black trees	✓	*balance*

An interesting found invariant

- Acyclicity expressed as size properties in the Singly Linked Lists case study

```
thiz.size = # (thiz.header.*next - Null)
  and
not (thiz.size = #(thiz.header.*next))
```

Future Work

- Search for more general specifications patterns to consider in catalogs

- Develop more complex genetic operators

Questions?

Thank you :)