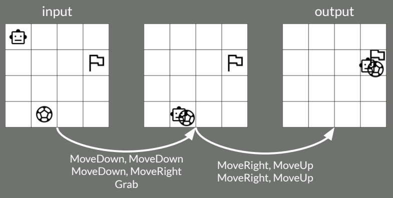
EVOLVING A LANGUAGE FOR PROGRAM SYNTHESIS

1 BACKGROUND: PROGRAM SYNTHESIS

"Automatically finding a program in the underlying programming language that satisfies the user intent."

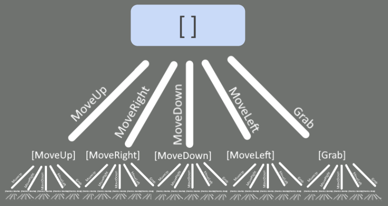
PROGRAMMING BY EXAMPLE

Task: bring the ball to the flag



Optimal program: [MoveDown, MoveDown, MoveDown, MoveRight, Grab, MoveRight, MoveUp, MoveRight, MoveUp]

VAST SEARCH SPACE



With 10 language predicates & creating a program of length 10 = 10.000.000.000 potential programs

3 METHODOLOGY: GENETIC ALGORITHM

"Uses biologically inspired operators to generate high-quality solutions to optimisation problems by evolving a population of chromosomes over many generations."

CHROMOSOME

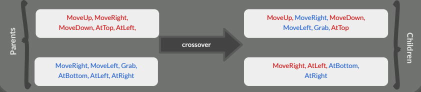
Domain-Specific Language. For example: MoveUp, MoveRight, MoveDown, MoveLeft, Grab

FITNESS

Calculated by using the chromosome as the language for a set of program synthesis tasks

$$fitness = \frac{percentage\_solved}{average\_time\_taken}$$

CROSSOVER


Parents
MoveUp, MoveRight, MoveDown, AtTop, AtLeft,
MoveRight, MoveLeft, Grab, AtBottom, AtLeft, AtRight
crossover
MoveUp, MoveRight, MoveDown, MoveLeft, Grab, AtTop
MoveRight, AtLeft, AtBottom, AtRight
Children

MUTATION

Unnecessary predicates only lead to increase in search time

MoveUp, MoveDown, MoveLeft, Grab, ~~AtTop~~ → remove → MoveUp, MoveDown, MoveLeft, Grab

Some predicates are necessary to solve more tasks

MoveUp, MoveDown, MoveLeft, AtTop → add → MoveUp, MoveDown, MoveLeft, AtTop, Grab

Some composite predicates are created frequently to solve tasks, adding these to a language may decrease its search time

We consider these composite predicates based on how frequently they appear in a successful program:

[LoopWhileThen (NotAtBottom [MoveDown], [Grab]), MoveUp, MoveRight]

MoveUp, MoveRight, MoveDown, MoveLeft, Grab, AtTop

add composite

MoveUp, MoveRight, MoveDown, MoveLeft, Grab, AtTop, LoopWhileThen (NotAtBottom [MoveDown], [Grab]),

2 RESEARCH QUESTIONS

MOTIVATION

1) The domain-specific language heavily influences the size of the search space
2) The size of the search space heavily influences the program synthesis time
3) Designing a domain-specific language is a complex optimisation problem
4) Genetic algorithms have proven to be useful for complex optimisation problems

QUESTIONS

Main question

"Can we evolve a programming language to speed up program synthesis?"

Research questions

1. How can we translate a DSL into a chromosome?
2. How can we add composite predicates to a DSL?
3. How can we use genetic programming techniques to evolve a DSL?
4. How does a program synthesiser using an evolved DSL compare to one using the standard DSL?

4 RESULTS

THREE DOMAINS

Program synthesiser finds a program that transforms the input into the output.

Input = the state the program starts at.
Output = the desired final state.

string domain

Task: to transform a string into a derived string.

Input: a string.
Output: a different string.

robot domain

Task: to pick up a ball and bring it to the flag.

Input: a grid with a robot, ball, and flag.
Output: a grid with the robot and ball at the flag.

pixel domain

Task: to copy a given drawing.

Input: blank canvas.
Output: canvas with the copied drawing.

EVOLVED DOMAIN-SPECIFIC LANGUAGES

string

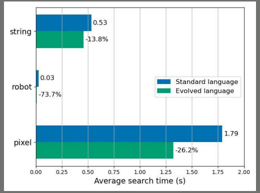
MoveRight, MoveLeft, MakeUppercase, MakeLowercase, AtEnd, NotAtEnd, NotAtStart, IsLetter, IsNotLetter, IsUppercase, IsNotUppercase, IsNumber, IsNotNumber, Drop

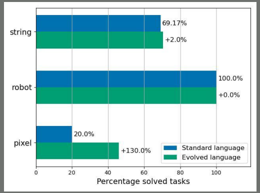
robot

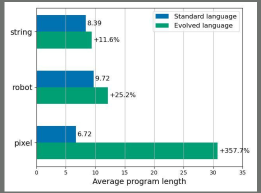
MoveUp, MoveRight, MoveDown, MoveLeft, Drop, Grab

pixel

MoveUp, MoveRight, MoveDown, MoveLeft, Draw







5 CONCLUSION

WHAT IS SHOWN

- Evolved languages speed up program synthesis in each domain considerably
- Using evolved languages solved same number or more tasks
- Evolved languages all had less predicates than original counterparts
- Adding composite predicates to the language is not worth it, even when only evolving for the 10% most complex tasks
- For domains robot and pixel it is not worth it to be able to create composite tokens

6 FUTURE WORK

SUGGESTIONS

- For some domains, some predicates are necessary (e.g. 'draw' when the task is to draw). These predicates should be harder to remove and easier to add, could be based on how often predicates appear in successful programs.
- Some composite predicate types are never found in successful programs. Could consider to include these types in chromosomes, because they might be useless and they contribute very heavily to search space size.

Research done by Philip Tempelman (p.i.tempelman@student.tudelft.nl), supervised by Sebastijan Dumančić (s.dumancic@tudelft.nl) for Delft University of Technology BSc Computer Science & Engineering Bachelor Thesis