Alternation as a Programming Paradigm[◇]

PPDP'09, Coimbra, Portugal

Wolfgang Dvořák[†], Georg Gottlob[*], Reinhard Pichler[†], Stefan Woltran[†]

[†] Institut für Informationssysteme
Technische Universität Wien

[*] Computing Laboratory
Oxford University

September 7, 2009

[◇] This work was supported by the Austrian Science Fund (FWF), project P20704-N18.

Motivation

Why considering Alternation as a Programming Paradigm ?

Motivation

Why considering Alternation as a Programming Paradigm ?

- Alternating (Turing) Machines are a well-known concept in computational complexity theory.
- Alternating algorithms are often used to prove containment of a problem in a particular complexity class.
- There are feasible alternating complexity classes and problems.
- Many practical problems have natural descriptions in terms of alternation.

Motivation

Why considering Alternation as a Programming Paradigm ?

- Alternating (Turing) Machines are a well-known concept in computational complexity theory.
- Alternating algorithms are often used to prove containment of a problem in a particular complexity class.
- There are feasible alternating complexity classes and problems.
- Many practical problems have natural descriptions in terms of alternation.

There is no support of alternation in the common programming languages

Motivation

Integration of declarative language constructs in imperative languages has been an important goal in many research projects.

Some previous projects that integrated non-determinism in imperative languages:

- Alma-0 (A.R. Apt and A. Schaerf): Integrating non-determinism in Modula-2
- Paslog (A. Radensky): Horn-clause programming in Pascal

Outline

Computation Modes

In theoretical computer science there are different modes of computation

Computation Modes

In theoretical computer science there are different modes of computation

deterministic machines

- There is exactly one possible successor for each configuration of the machine.
- A computation for an input is simply a *sequence* of configurations.
- The value of a computation is given by the last configuration in the sequence.

Computation Modes

In theoretical computer science there are different modes of computation

deterministic machines

- There is exactly one possible successor for each configuration of the machine.
- A computation for an input is simply a *sequence* of configurations.
- The value of a computation is given by the last configuration in the sequence.

nondeterministic machines

- For each configuration of the machine there may be many possible successors.
- The possible computations for an input can be represented as a *computation tree*.
- It is not clear what is the output for a given input.

Nondeterministic Acceptance

(ordinary) nondeterministic acceptance

A machine accepts an input if at least one possible computation accepts.

Nondeterministic Acceptance

(ordinary) nondeterministic acceptance

A machine accepts an input if at least one possible computation accepts.

(co-) nondeterministic acceptance

A machine accepts an input iff all possible computations accept.

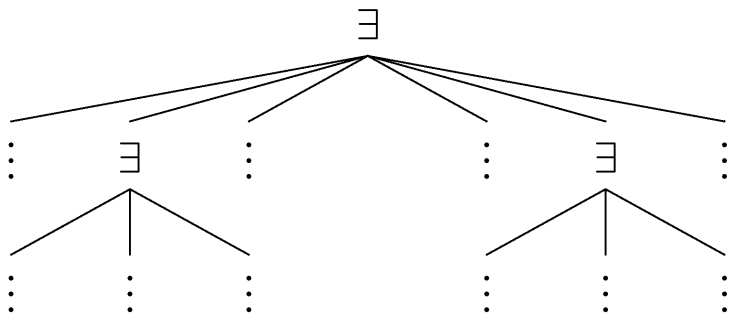
Nondeterministic Acceptance

(ordinary) nondeterministic acceptance

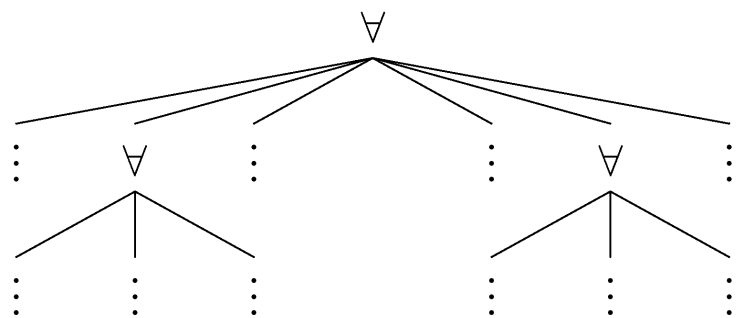
A machine accepts an input if at least one possible computation accepts.

(co-) nondeterministic acceptance

A machine accepts an input iff all possible computations accept.



NP



co-NP

Alternation

Alternating machines combine the existential acceptance from NP with the universal acceptance from co-NP

alternating machines

An alternating machine is a nondeterministic machine with two kinds of configurations:

- universal configurations: evaluate to true iff all successors evaluate to true
- existential configurations: evaluate to true iff at least one successor evaluates to true

The machine accepts an input iff the initial configuration evaluates to true.

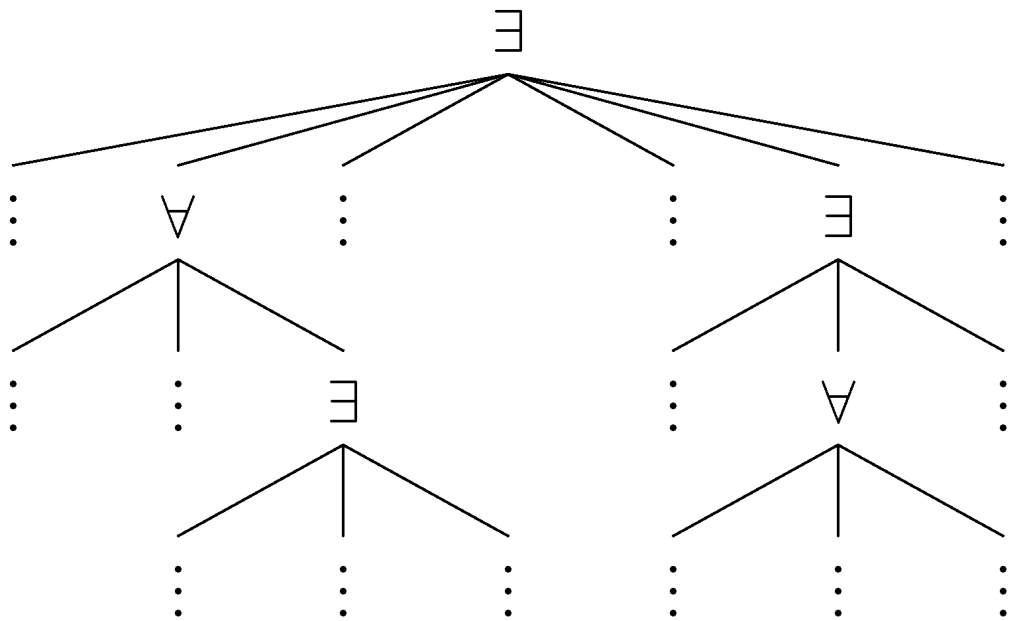


Figure: alternating acceptance mode

Alternation

- The class of problems decidable by an Alternating Turing Machine in logarithmic space corresponds to the class of polynomial time decidable problems on deterministic Turing Machines

$$\text{ALOGSPACE} = \text{P}$$

Alternation

- The class of problems decidable by an Alternating Turing Machine in logarithmic space corresponds to the class of polynomial time decidable problems on deterministic Turing Machines

  $$\text{ALOGSPACE} = \text{P}$$

- There are many problems with natural specifications in terms of alternation (even in ALOGSPACE).
  e.g. Horn Minimal Model/Satisfiability, Winning Strategies in Games, Circuit Evaluation, Alternating Graph Accessibility, ...

Example

Acyclic Geography Game (AGG)

instance: An directed acyclic graph $G = (V, E)$ and a vertex s.
The game is played as follows:
We start with a token on the vertex s. Player 1 has the first move and then the players alternate moving. In each move the active player can move the token along one edge. The first player with no possible move loses.
question: Is there a winning strategy for Player 1?

Example

Acyclic Geography Game (AGG)

instance: An directed acyclic graph $G = (V, E)$ and a vertex s.
The game is played as follows:
We start with a token on the vertex s. Player 1 has the first move and then the players alternate moving. In each move the active player can move the token along one edge. The first player with no possible move loses.
question: Is there a winning strategy for Player 1?

alternating approach:

Player 1 has a winning strategy starting in s if there EXISTS an edge (s, v) such that FORALL edges (v, v') there is winning strategy for player 1 starting in v'.

Alter-Java

Alter-Java is a language extension of the imperative part of Java which adds constructs for alternation.

The main concepts of the Alter-Java language:

- *input variables* - input for the computation, must not be modified during the computation
- *work variables* - specifying the configurations of the program
- the program is written in *states* - a state computes successor states and combines their results to a return value
- the program starts in an initial configuration. The output of the program is the result of the initial configuration

Alter-Java

Additional keywords in the Alter-Java language:

- *atm* - at the beginning of an Alter-Java program. Followed by the program´s name
- *state* - defines a state of the program. Followed by the name of a state and a block with the code of the state
- *forall*, *exists* - declares the acceptance mode of a configuration. Followed by a block defining the successor configurations
- *accept*, *reject* - immediate acceptance / rejection of the configuration

Acyclic Geographic program

```
package at.ac.tuwien.dbai.alternation.examples;

atm Geographic (Graph<Position> gameBoard,
                Graph.Node<Position> startNode)
{
    Graph.Node<Position> position=startNode;

    state Player1 {
        [...]
    }

    state Player2 {
        [...]
    }
}
```

state Player1

```
state Player1 {
    exists {
        for (Graph.Node<Position> node : position.
            getChildren()){
            Player2{
                position=node;
            }
        }
    }
}
```

state Player2

```
state Player2 {
    forall {
        for (Graph.Node<Position> node : position.
            getChildren()){
            Player1{
                position=node;
            }
        }
    }
}
```

```
package at.ac.tuwien.dbai.alternation.examples;

atm Geographic (Graph<Position> gameBoard,
                Graph.Node<Position> startNode)
{
    Graph.Node<Position> position=startNode;

    state Player1 {
        exists {
            for (Graph.Node<Position> node : position.getChildren())
                {
                Player2{
                    position=node;
                }
            }
        }
    }

    state Player2 {
        forall {
            for (Graph.Node<Position> node : position.getChildren())
                {
                Player1{
                    position=node;
                }
            }
        }
    }
}
```

Implementation

Alter-Java Implementation:

- A compiler transforming Alter-Java to Java.
- A framework executing alternating Programs.

Framework:

- Computation trees of ATMs grow exponentially with the input (even for ALOGSPACE).
  $\hookrightarrow$ naive tree traversal algorithms are not feasible.
- Our framework uses a tabled evaluation algorithm to provide polynomial runtime.
- Our framework offers cycle detection to handle infinite computation paths.

Experimental Results

First experiments with the following problems:

- Two Player Games
- Definite Horn Minimal Model

Experience:

- Comparable runtime to traditional Java programs
- Lines of code: significant reduction
  $\hookrightarrow$ example 700 loc in Java vs 100 loc in Alter-Java
- Cycle-detection is critical for many applications

Conclusion

The main contributions of our paper:

- Examples of feasible alternating problems and programs.
- Alter-Java: A language offering language constructs for expressing alternation.
- Alter-Java-Implementation: A framework executing Alter-Java programs in feasible time and space bounds.
- Experiments confirming the feasibility of Alter-Java.

Conclusion

The main contributions of our paper:

- Examples of feasible alternating problems and programs.
- Alter-Java: A language offering language constructs for expressing alternation.
- Alter-Java-Implementation: A framework executing Alter-Java programs in feasible time and space bounds.
- Experiments confirming the feasibility of Alter-Java.

Future work:

- Explore the potential of alternating programs for LOGCFL-problems.
- Identifying possible applications of alternation in the field of Computer Aided Verification (e.g. CTL).