MiniZinc Linter

Tool for Static Model Analysis

Erik Rimskog

Uppsala University

Outline

Outline

- MiniZinc
- Linter
- Rules
- Implementation
- Summary

MiniZinc, Tool for Solving Problems

What it is

Toolchain for modelling combinatorial optimisation problems

MiniZinc, Tool for Solving Problems

What it is

Toolchain for modelling combinatorial optimisation problems using *decision variables* (e.g. x) and *constraints* (e.g. $x \neq 4$).

MiniZinc, Tool for Solving Problems

What it is

Toolchain for modelling combinatorial optimisation problems using *decision variables* (e.g. x) and *constraints* (e.g. $x \neq 4$). It is solver independent.

MiniZinc, Tool for Solving Problems

What it is

Toolchain for modelling combinatorial optimisation problems using *decision variables* (e.g. x) and *constraints* (e.g. $x \neq 4$). It is solver independent.

Travelling Salesman Problem

Find an order to visit $n \in \mathbb{N}^+$ cities in to minimise the total distance travelled.

Outline

Linters in General

What a linter is

A tool to perform static analysis on source code. It gives suggestions, okay to be wrong.

Linters in General

What a linter is

A tool to perform static analysis on source code. It gives suggestions, okay to be wrong.

- Error-prone constructs

Linters in General

What a linter is

A tool to perform static analysis on source code. It gives suggestions, okay to be wrong.

- Error-prone constructs
- Style

Linters in General

What a linter is

A tool to perform static analysis on source code. It gives suggestions, okay to be wrong.

- Error-prone constructs
- Style
- Performance

Linters in General

What a linter is

A tool to perform static analysis on source code. It gives suggestions, okay to be wrong.

- Error-prone constructs
- Style
- Performance
- etc.

Linters in General

What a linter is

A tool to perform static analysis on source code. It gives suggestions, okay to be wrong.

- Error-prone constructs
- Style
- Performance
- etc.

Many statements can't be proven!

Linters in General

What a linter is

A tool to perform static analysis on source code. It gives suggestions, okay to be wrong.

- Error-prone constructs
- Style
- Performance
- etc.

Many statements can't be proven!

Have to introduce limitations and approximations.

Linter Rules

Rule

A linter performs several checks, also called rules. Each rule can be disabled or enabled individually.

Demo

```
$ lzn nsp_1.mzn
```

Demo

```
$ lzn nsp_1.mzn
```

```
/home/erik/Documents/minizinc-benchmarks/nsp/nsp_1.mzn:67.1-67.55: possibly non-functionally defined variable not in search hint [non-func-hint(9)]
    |     array [period, shifts] of var int:           coverage;
    |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/erik/Documents/minizinc-benchmarks/nsp/nsp_1.mzn:66.1-66.62: possibly non-functionally defined variable not in search hint [non-func-hint(9)]
    |     array [nurses, period] of var shifts_and_off:  nurses_schedule;
    |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/erik/Documents/minizinc-benchmarks/nsp/nsp_1.mzn:67.1-67.55: no explicit domain on variable declaration [unbounded-variable(13)]
    |     array [period, shifts] of var int:           coverage;
    |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/erik/Documents/minizinc-benchmarks/nsp/nsp_1.mzn:68.1-68.60: no explicit domain on variable declaration [unbounded-variable(13)]
    |     array [shifts] of var int:                   shifts_values;
    |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/erik/Documents/minizinc-benchmarks/nsp/nsp_1.mzn:68.1-68.60: is only constrained to par values, shouldn't be var [constant-variable(4)]
    |     array [shifts] of var int:                   shifts_values;
    |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/erik/Documents/minizinc-benchmarks/nsp/nsp_1.mzn:88.13-88.28: constrained here
    ^     shifts_values[j] = j
    |     ^~~~~~~~~~~~~~~
```

Outline

Rule: Unused Variables and Functions

Motivation

Unused variables and functions don't contribute to the model and should be removed.

Rule: Unused Variables and Functions

Motivation

Unused variables and functions don't contribute to the model and should be removed. Being mentioned inside constraints or other used functions counts as usage.

Rule: Unused Variables and Functions

Motivation

Unused variables and functions don't contribute to the model and should be removed. Being mentioned inside constraints or other used functions counts as usage.

Example

```
int: K = 69;
var 1..K: a;        % uses K
var int: b;         % unused
constraint a > 2; % uses a, and indirectly K
solve satisfy;
```

Rule: Unused Variables and Functions Cont.

Steps

1. Find all variables and functions **2.** Calculate dependencies **3.** Find all uses and recursively mark nodes as used

Rule: Unused Variables and Functions Cont.

Steps

1. Find all variables and functions **2.** Calculate dependencies **3.** Find all uses and recursively mark nodes as used

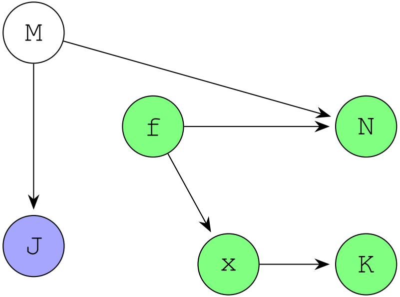
```
int: K = 1;
int: N = 1;
int: M = let {int: J = 5} in J+N;
var 0..K: x;
function var int: f() = x+N;
solve minimize f();
```

Rule: Unused Variables and Functions Cont.

Steps

1. Find all variables and functions **2.** Calculate dependencies **3.** Find all uses and recursively mark nodes as used

```
int: K = 1;
int: N = 1;
int: M = let {int: J = 5} in J+N;
var 0..K: x;
function var int: f() = x+N;
solve minimize f();
```

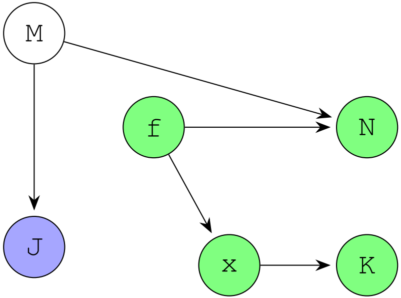


Rule: Unused Variables and Functions Cont.

Steps

1. Find all variables and functions **2.** Calculate dependencies **3.** Find all uses and recursively mark nodes as used **4.** Simplify output

```
int: K = 1;
int: N = 1;
int: M = let {int: J = 5} in J+N;
var 0..K: x;
function var int: f() = x+N;
solve minimize f();
```



Rule: Missing Domain on Decision Variables

Motivation

Decision variables should always have tight domain to limit the amount of potential values.

Rule: Missing Domain on Decision Variables

Motivation

Decision variables should always have tight domain to limit the amount of potential values.

Bad

```
var int: a;
```

Good

```
var 1..5: a;
```

Rule: Missing Domain on Decision Variables

Motivation

Decision variables should always have tight domain to limit the amount of potential values.

Bad

```
var int: a;
```

Good

```
var 1..5: a;
var int: b = a+1;
```

Rule: Missing Domain on Decision Variables

Motivation

Decision variables should always have tight domain to limit the amount of potential values.

Bad

```
var int: a;
```

Good

```
var 1..5: a;
var int: b = a+1;
constraint b = a+1;
```

Rule: Missing Domain on Decision Variables Cont.

Steps

1. Find all decision variables
2. Search constraints for equalities (`a=...`)

Rule: Missing Domain on Decision Variables Cont.

Steps

1. Find all decision variables
2. Search constraints for equalities (`a=...`)

Will *not* always constrain `a`

```
constraint ... -> a=2;
constraint a=2 \/ ...;
```

Rule: Missing Domain on Decision Variables Cont.

Steps

1. Find all decision variables
2. Search constraints for equalities (`a=...`)

Will *not* always constrain `a`

```
constraint ... -> a=2;
constraint a=2 \/ ...;
```

Will always constrain `a`

```
constraint a=2;
```

Rule: Missing Domain on Decision Variables Cont.

Steps

1. Find all decision variables
2. Search constraints for equalities (`a=...`)

Will *not* always constrain `a`

```
constraint ... -> a=2;
constraint a=2 \/ ...;
```

Will always constrain `a`

```
constraint a=2;
constraint a=2 /\ ...;
```

Rule: Missing Domain on Decision Variables Cont.

Steps

1. Find all decision variables
2. Search constraints for equalities (`a=...`)

Will *not* always constrain `a`

```
constraint ... -> a=2;
constraint a=2 \/ ...;
```

Will always constrain `a`

```
constraint a=2;
constraint a=2 /\ ...;
constraint forall([a=2]);
```

General Limitations

Obfuscate the model

```
constraint a=2 \/ false;
```

General Limitations

Obfuscate the model

```
constraint a=2 \/ false; % linter won't find
```

General Limitations

Obfuscate the model

```
constraint a=2 \/ false; % linter won't find
constraint true /\ a=2;
```

General Limitations

Obfuscate the model

```
constraint a=2 \/ false; % linter won't find
constraint true /\ a=2;
constraint [a] = [2];
```

General Limitations

Obfuscate the model

```
constraint a=2 \/ false; % linter won't find
constraint true /\ a=2;
constraint [a] = [2];
```

Not feasible to account for all cases

They are too many ways to obfuscate! Maybe add rules to remove some obfuscation?

General Limitations Cont.

No reliance on instance variables (par)

General Limitations Cont.

No reliance on instance variables (par)

Determine whether all individual elements have been accessed in an array

General Limitations Cont.

No reliance on instance variables (par)

Determine whether all individual elements have been accessed in an array

```
int: A = 5;
array[1..A] of var int: xs;
constraint forall(i in 1..5)(xs[i] = ...);
```

General Limitations Cont.

No reliance on instance variables (par)

Determine whether all individual elements have been accessed in an array

```
int: A = 5;
array[1..A] of var int: xs;
constraint forall(i in 1..5)(xs[i] = ...);
```

The index sets have to be exactly the same.

Outline

Abstract Syntax Tree

Parser

Abstract Syntax Tree

Parser

Reads text and structures it into a data structure that can be processed, AST is a common one.

Abstract Syntax Tree

Parser

Reads text and structures it into a data structure that can be processed, AST is a common one.

MiniZinc's parser was reused

- Easier to maintain one than two
- Faster development
- Use existing functionality like the typechecker

Abstract Syntax Tree

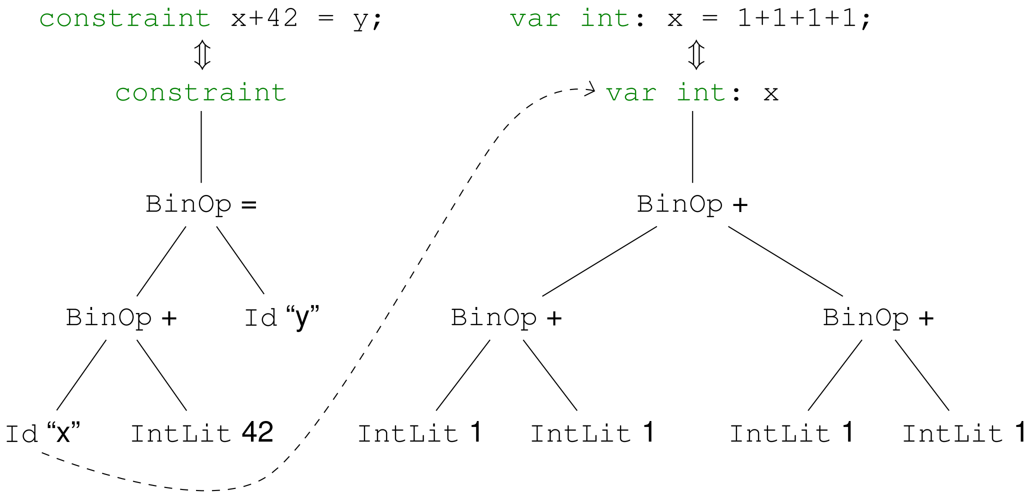
Parser

Reads text and structures it into a data structure that can be processed, AST is a common one.

MiniZinc's parser was reused

- Easier to maintain one than two
- Faster development
- Use existing functionality like the typechecker

MiniZinc AST



AST Searcher

How to Search

Find locations of interest in an AST from a path specification (regex-like).

AST Searcher

How to Search

Find locations of interest in an AST from a path specification (regex-like).

- $+$
- Find an addition node

AST Searcher

How to Search

Find locations of interest in an AST from a path specification (regex-like).

- `+`
- `+ >> int`

- Find an addition node
- Find an addition node with an integer under it

AST Searcher

How to Search

Find locations of interest in an AST from a path specification (regex-like).

- `+`
- `+ >> int`
- `= > id`

- Find an addition node
- Find an addition node with an integer under it
- Find id directly under equals node

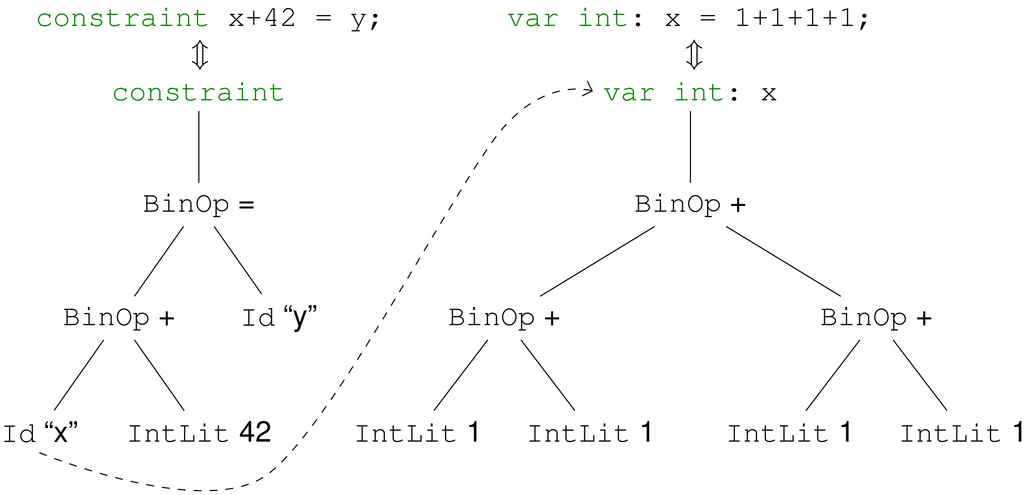
AST Searcher

How to Search

Find locations of interest in an AST from a path specification (regex-like).

- `+` — Find an addition node
- `+ >> int` — Find an addition node with an integer under it
- `= > id` — Find id directly under equals node
- `let > = >> + > id` — Can be arbitrarily long

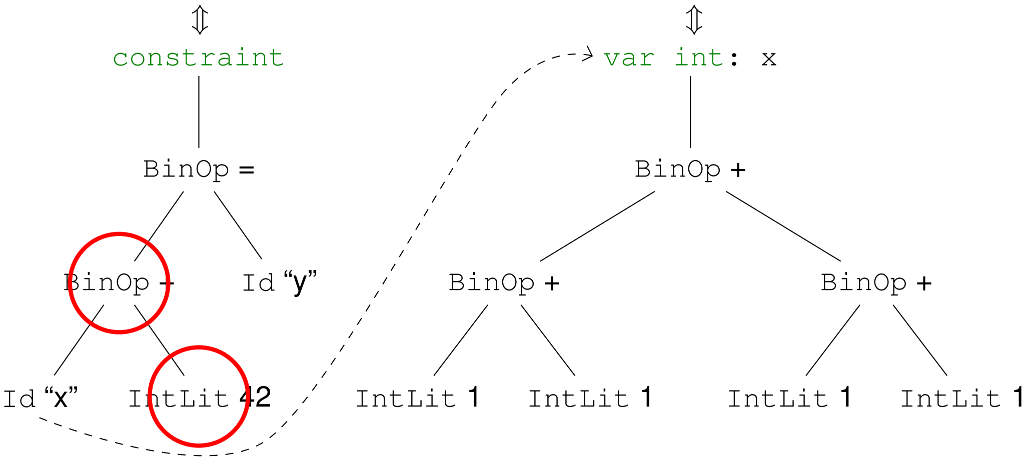
MiniZinc AST



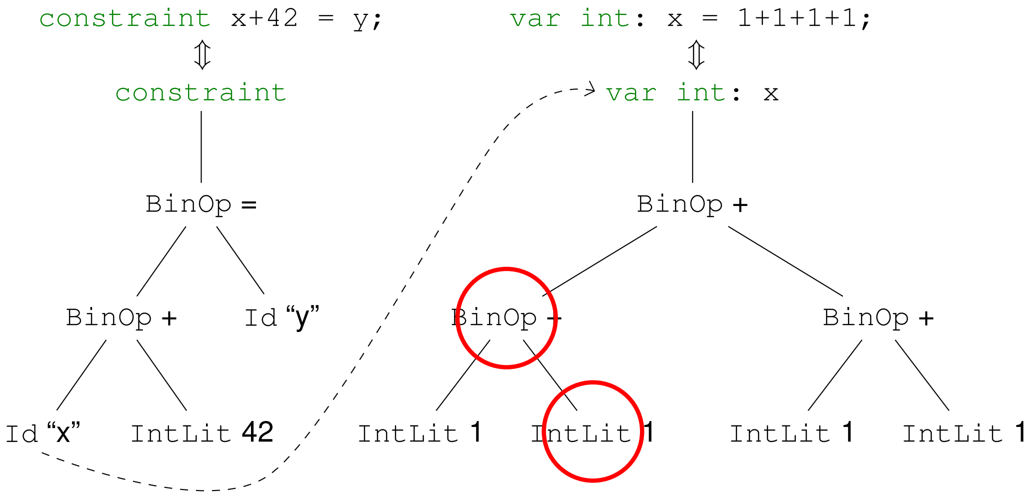
MiniZinc AST

```
constraint x+42 = y;
```

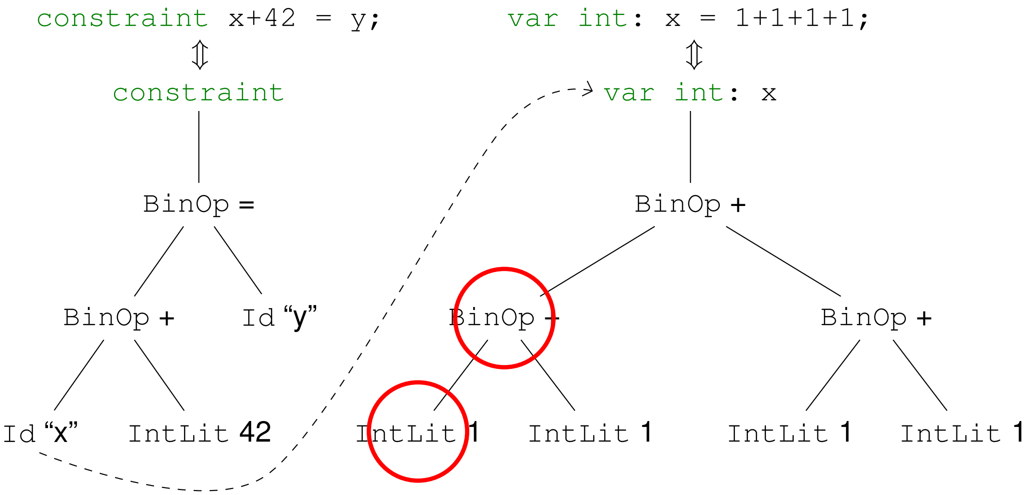
```
var int: x = 1+1+1+1;
```



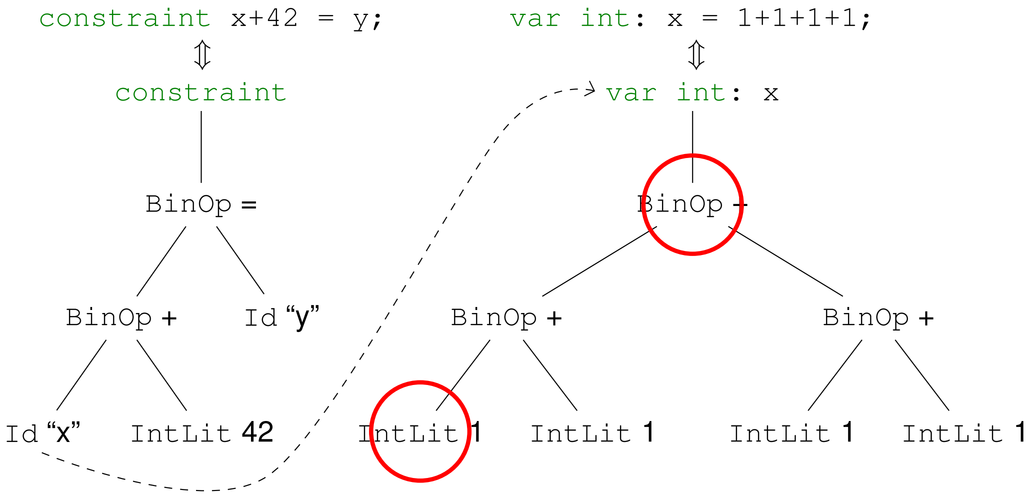
MiniZinc AST



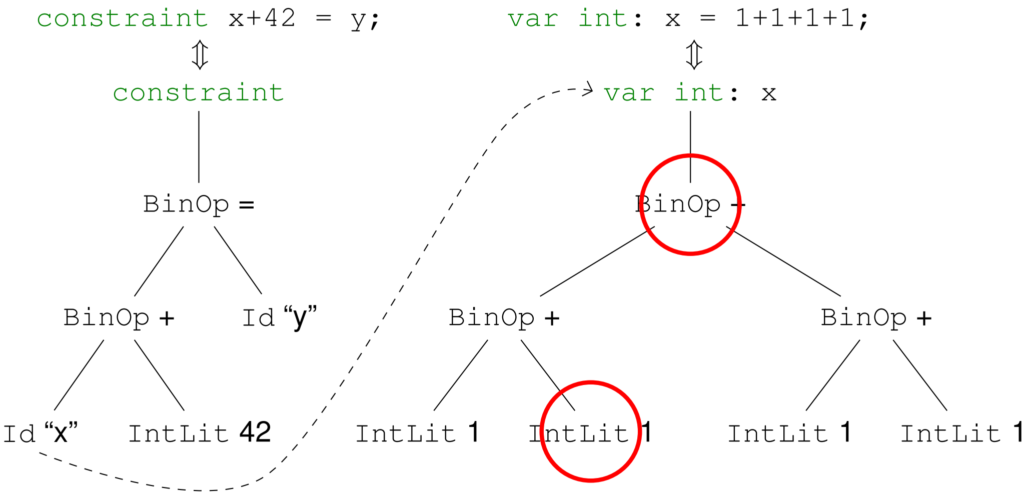
MiniZinc AST



MiniZinc AST



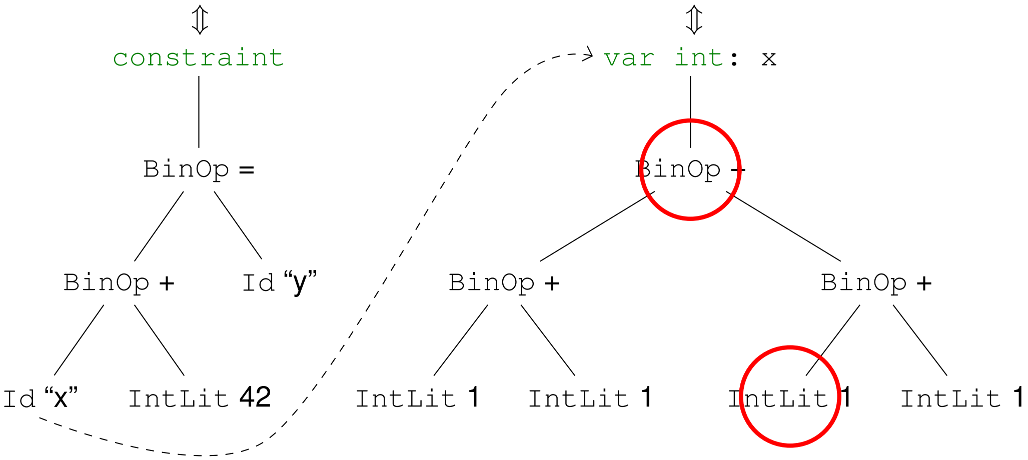
MiniZinc AST



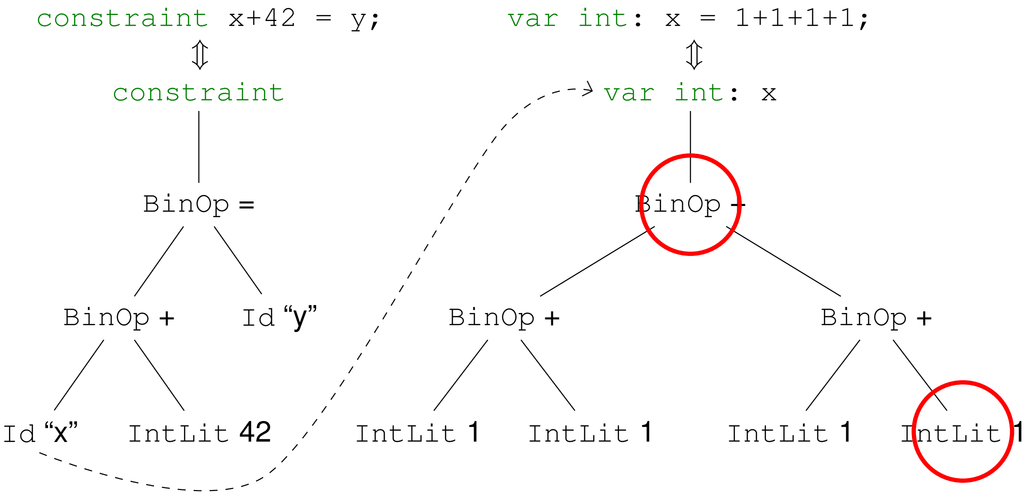
MiniZinc AST

`constraint x+42 = y;`

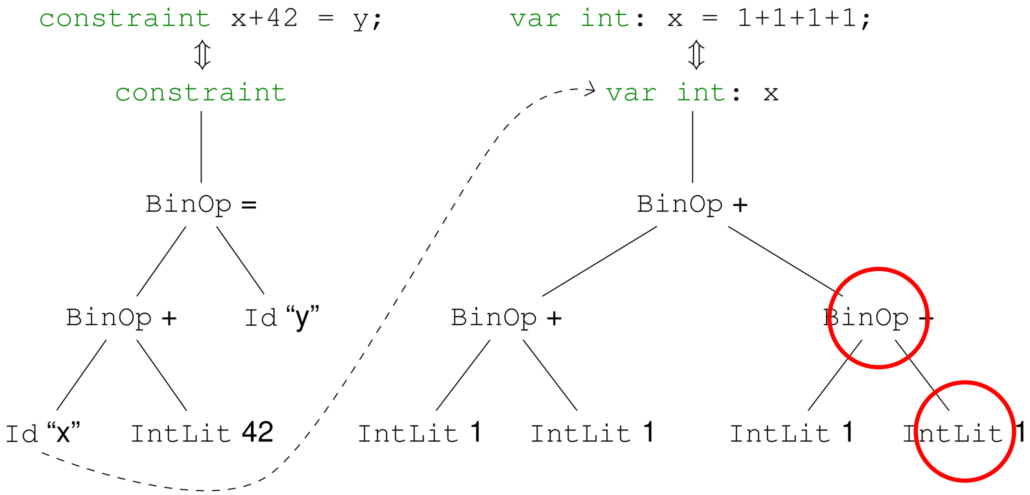
`var int: x = 1+1+1+1;`



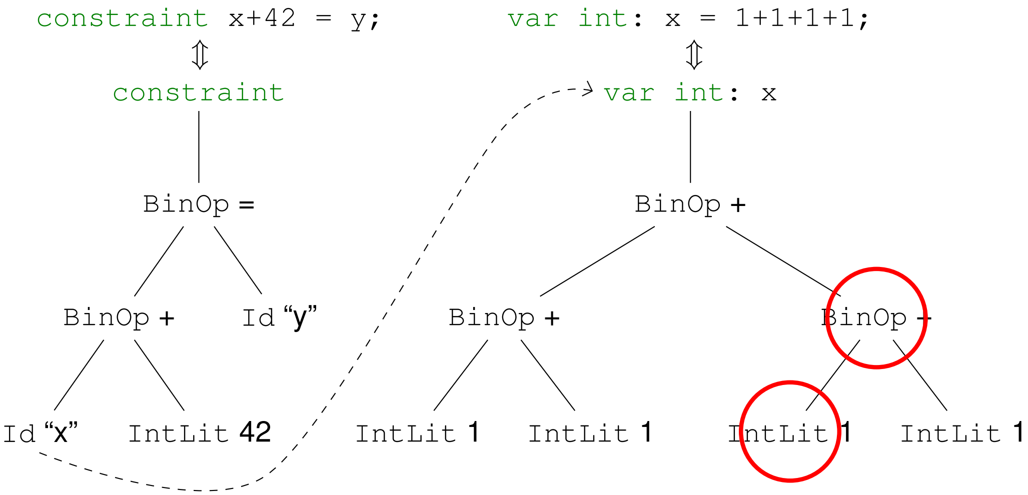
MiniZinc AST



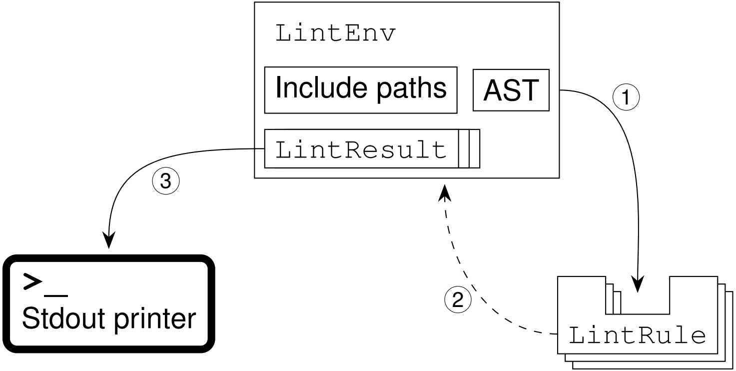
MiniZinc AST



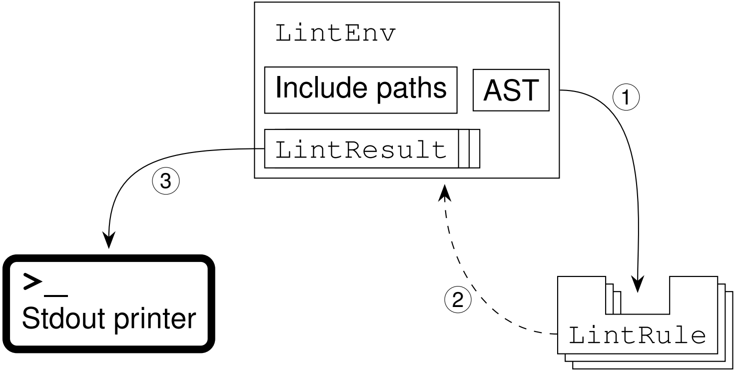
MiniZinc AST



Execution Overview

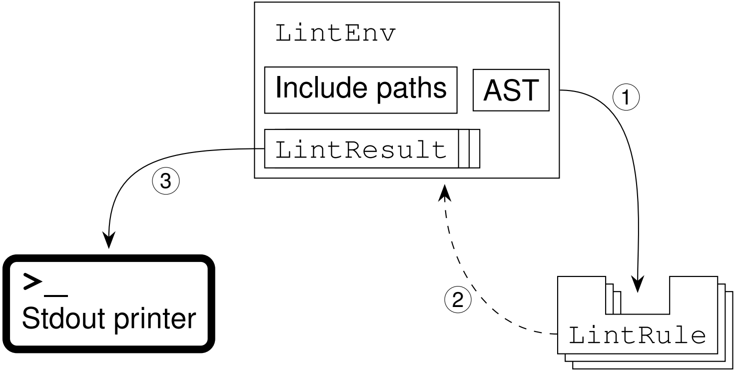


Execution Overview



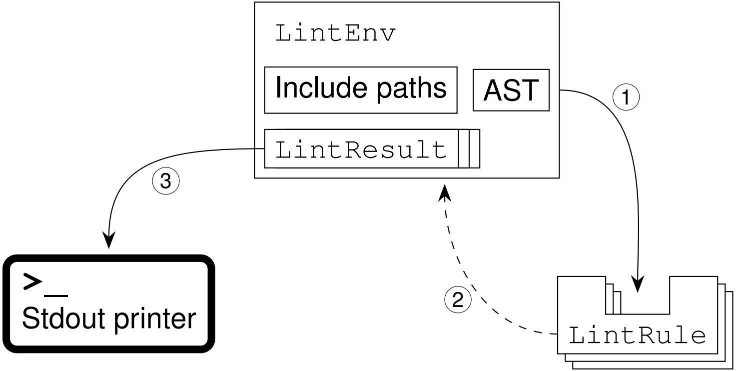
1. Each rule reads the AST from the environment

Execution Overview



1. Each rule reads the AST from the environment
2. They write back results

Execution Overview



1. Each rule reads the AST from the environment
2. They write back results
3. The results are displayed in some way

Outline

Summary

- MiniZinc is a modelling language that supports several different solvers.
- Decision variables are unknows and constraints constrain them.
- A linter for static analysis of models was created.
- Some things are impossible to deduce.
- Written in C++.
- The MiniZinc parser was reused.
- An AST searcher searches for locations of interest in models.