

CONSTRAINED ZIG

Exploring properties of constraint programs

Lukas Pietzschmann

lukas.pietzschmann@uni-ulm.de

Zigtoberfest

Hochschule München

October 19th, 2024



ABOUT ME



Hi, I'm Lukas!



► Studies



I'm studying computer science for my master's degree at Ulm University. Currently, I'm working on my thesis.

► Interests



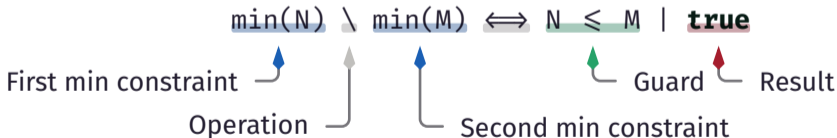
I enjoy various things, but compilers, typesetting, and functional programming spark the most joy inside me.

2

MOTIVATION

A first small example

Constraint Programming represents one of the closest approaches computer science has yet made to the holy grail of programming: the user states the problem, the computer solves it.
(Eugene C. Freuder [Fre97])



3.1

INTRODUCTION TO CHR

CHR knows about three kinds of rules:

Propagation

$$C \Rightarrow G \mid CN$$

CN is inferred if G holds

Simplification


$$C \iff G \mid CN$$

CO is simplified to CN if G holds

Simpagation

$$C_1 \setminus C_2 \iff G \mid CN$$

C_2 is simplified to CN if G holds

C	Head	CHR Constraints
G	Guard	Conjunction of <u>built-ins</u>  Zig code
CN	Body	CHR Constraints and conjunction of built-ins

CHR knows about three kinds of rules:

Generalized Simpagation Rule

Behind the curtain, the embedding represents every rule as a 4-tuple:

$$\langle KH, RH, G, B \rangle$$

KH: Kept Head *RH*: Removed Head *G*: Guard *B*: Body

A propagation rule would then look like this: $\langle KH, \emptyset, G, B \rangle$



3.2

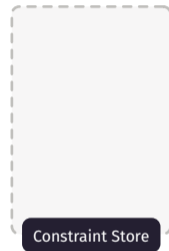
INTRODUCTION TO CHR

Constraint Store

› Query ...

$\text{min}(N) \setminus \text{min}(M) \iff N \leq M \mid \mathbf{true}$

› Add constraints to solve for



3.2

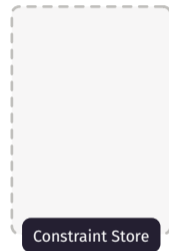
INTRODUCTION TO CHR

Constraint Store

```
> min(3)
```

```
min(N) \ min(M)  $\iff$  N  $\leq$  M | true
```

> Add constraints to solve for



3.2

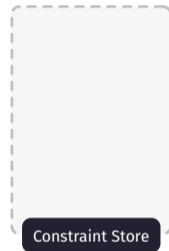
INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1)
```

```
min(N) \ min(M)  $\iff$  N  $\leq$  M | true
```

> Add constraints to solve for



3.2

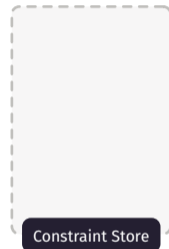
INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

```
min(N) \ min(M)  $\iff$  N  $\leq$  M | true
```

> Add constraints to solve for



3.2

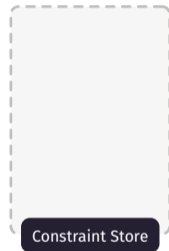
INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

$\text{min}(N) \ \backslash \ \text{min}(M) \iff N \leq M \mid \mathbf{true}$

› Check if we can find a matching for the currently active constraint



3.2

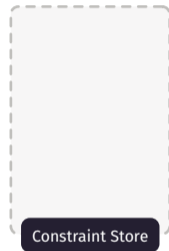
INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

$\text{min}(3) \setminus \text{min}(M) \iff 3 \leq M \mid \text{true}$

› Check if we can find a matching for the currently active constraint



3.2

INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

$\text{min}(N) \setminus \text{min}(M) \iff N \leq M \mid \text{true}$

► Check if we can find a matching for the currently active constraint



3.2

INTRODUCTION TO CHR

Constraint Store

```
› min(3), min(1), min(5)
```

```
min(N) \ min(M)  $\iff$  N  $\leq$  M | true
```

› Fire the rule that matched

min(3)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
› min(3), min(1), min(5)
```

```
min(N) \ min(M)  $\iff$  N  $\leq$  M | true
```

› Make next query constraint active

min(3)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

$\text{min}(N) \setminus \text{min}(M) \iff N \leq M \mid \text{true}$

› Check if we can find a matching for the currently active constraint

min(3)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

$\text{min}(1) \setminus \text{min}(3) \iff 1 \leq 3 \mid \text{true}$

› Check if we can find a matching for the currently active constraint

min(3)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

$\text{min}(1) \setminus \text{min}(3) \iff 1 \leq 3 \mid \text{true}$

> Fire the rule that matched



3.2

INTRODUCTION TO CHR

Constraint Store

```
› min(3), min(1), min(5)
```

```
min(N) \ min(M)  $\iff$  N  $\leq$  M | true
```

› Fire the rule that matched

min(1)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
› min(3), min(1), min(5)
```

$\text{min}(N) \setminus \text{min}(M) \iff N \leq M \mid \mathbf{true}$

› Make next query constraint active

min(1)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

$\text{min}(N) \setminus \text{min}(M) \iff N \leq M \mid \text{true}$

› Check if we can find a matching for the currently active constraint

min(1)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
> min(3), min(1), min(5)
```

$\text{min}(1) \setminus \text{min}(5) \iff 1 \leq 5 \mid \text{true}$

› Check if we can find a matching for the currently active constraint

min(1)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
› min(3), min(1), min(5)
```

$\text{min}(N) \setminus \text{min}(M) \iff N \leq M \mid \mathbf{true}$

› Make next query constraint active

min(1)

Constraint Store

3.2

INTRODUCTION TO CHR

Constraint Store

```
› min(3), min(1), min(5)
```

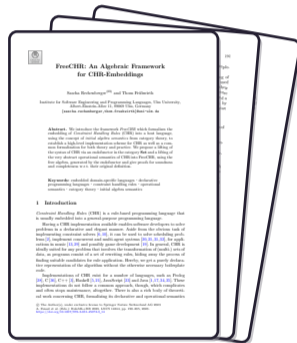
$\text{min}(N) \setminus \text{min}(M) \iff N \leq M \mid \mathbf{true}$

› The remaining constraints in the store are the solution

min(1)

Constraint Store

- Based on the concepts established in “FreeCHR: An Algebraic Framework for CHR-Embeddings” [RF23]
- Every rule is represented as a 4-tuple (see slide 3)
- We then compose more complex programs from subprograms — or ultimately from single rules
- Lastly, we apply the composition to a state until a fixpoint is reached



4.2

ZIGCHR ONLINE

- The embedding's source code is available on [GitHub](#)
- Clone it and try it out yourself!
- Also, the repo contains some of the examples we're discussing today



<https://lukas.pietzschmann.org/talks/zigtoberfest>



5.1

ANYTIME ALGORITHMS

- We can interrupt the execution at anytime
- The intermediate state will be an approximation of the final result
- After that, the program will continue like nothing happened
- This is especially useful if we need to guarantee a certain response time

5.2

EASY MEMORIZATION

- › Quiz: Change three symbols to make this have linear complexity

`fib(0, M) ⇔ M = 1`

`fib(1, M) ⇔ M = 1`

`fib(N, M) ⇔ N ≥ 2 |`

`fib(N-1, M1), fib(N-2, M2), M is M1 + M2`

`fib(0, M) ⇒ M = 1`

`fib(1, M) ⇒ M = 1`

`fib(N, M) ⇒ N ≥ 2 |`

`fib(N-1, M1), fib(N-2, M2), M is M1 + M2`

- We have to change the simplification rules to propagations
- This way, we do not remove calculated values from the store
- But we remember them for future use

Rule order matters

`throw(Coin) \iff head`
`throw(Coin) \iff tail`

- Depending on which rule we check first, we get a different result
- That's great for coin tossing
- Not so much for determinism

Constraint order matters

`set(K,V), c(K,V') \iff c(K,V)`

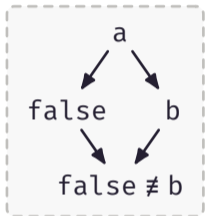
- Imagine the following query:
`c(x,0), set(x,1), set(x,2)`
- Depending on what constraint we select first, x will be set differently

► In confluent programs, the relation between the initial and final state is a function

5.4

CONFLUENCE

Achieve Confluence



- Both rules are applicable to a state containing only a
- From this, we can derive the two new states
- Confluence requires then to be joinable

$a \Rightarrow b$
 $a \Rightarrow \text{false}$

-
- We can make the two states joinable by adding a single rule

$a \Rightarrow b$
 $a \Rightarrow \text{false}$
 $b \Rightarrow \text{false}$

- ☑ Not every program can be made confluent

a \Rightarrow **true**

a \Rightarrow false

- ☑ Be aware of the semantics of your program

$\text{set}(K, V), c(K, V') \iff c(K, V)$

▲ does not equal ▼

$\text{set}(K, V), c(K, V') \iff V = V' \mid c(K, V)$

5.6 CONFLUENCE

Why we need it

- If a program terminates, we can check if it's confluent
- If it is, we get more cool properties for free:

Incrementality

- We can add constraints during the program's execution
- They will eventually participate in the computation
- The result will be the same as if was there from the beginning



Parallelism

- The algorithm can be executed in parallel
- Without any modifications

```

> map(square), reduce(add), v(2), v(3), v(4), v(5)

```

```

map(OP) \ v(X) ⇔
  C =.. [OP, X, R],
  call(C),
  r(R)

```

```

reduce(OP) \ r(X), r(Y) ⇔
  C =.. [OP, X, Y, R],
  call(C),
  r(R)

```

```

reduce(add)
map(square)
r(54)

```

Constraint Store

6

WHAT'S LEFT TO SAY



Constraint programming, too 😊

- [Fre97] Eugene C. Freuder. In: *Constraints 2.1* (1997), pp. 57–61. ISSN: 1383-7133. DOI: 10.1023/a:1009749006768. URL: <http://dx.doi.org/10.1023/A:1009749006768>.
- [Frü09] Thom Frühwirth. *Constraint handling rules*. Cambridge University Press, 2009.
- [RF23] Sascha Rechenberger and Thom Frühwirth. “FreeCHR: An Algebraic Framework for CHR-Embeddings”. In: *International Joint Conference on Rules and Reasoning*. Springer, 2023, pp. 190–205.

Lukas Pietzschmann

Munich, October 19th, 2024

lukas.pietzschmann@uni-ulm.de