Logical Rules for a Lexicographic Order Constraint Solver

Thom Frühwirth

Faculty of Computer Science
University of Ulm, Germany
www.informatik.uni-ulm.de/pm/mitarbeiter/fruehwirth/

CHR 2005, Barcelona, October 2005

Lexicographic Order

Many Applications

- Comparing and sorting, e.g. strings
- Termination analysis, e.g. rewrite systems
- Symmetry breaking in constraint programming
- Modelling preferences in constraint programming

Different algorithms for the `lex` constraint

- Non-incremental pointer-based imperative pseudo code, 45 lines
  Frisch/Hnich/Kiziltan et. al. CP 2002.
- Finite automaton based on 7 cases, 42 lines
  Carlsson/Beldiceanu, ESOP 2004.
- Simpler automata, interpreted, 16 lines
  Beldiceanu/Carlsson/Petit, CP 20004.

Non-intuitive non-logical code. Hard to analyse.

Constraint Handling Rules (CHR)

- **CHR:** Constraint programming language for Computational Logic
- Multi-headed guarded committed-choice rules
  transform multi-set of constraints until exhaustion
- Ideal for executable specifications and rapid prototyping
- Efficient: all algorithms implementable with optimal time complexity
- Incrementality and concurrency for free (on-line, any-time)
- Logical and operational semantics coincide strongly
- High-level supports semi-automatic program analysis + transformation:
  Confluence + completion, termination + time complexity, correctness...
- Implemenations in most Prolog systems, Java, Haskell
- 100s of applications from types, time tabling to cancer diagnosis

Use CHR for a global `lex` constraint?

Lexicographic Order

Definition

Given two sequences l_1 and l_2 of length n, $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$, then $l_1 \preceq_{lex} l_2$ iff $n=0$ or $x_1<y_1$ or $x_1=y_1$ and $[x_2, \ldots, x_n] \preceq_{lex} [y_2, \ldots, y_n]$.

Logical Specification

$$
\begin{array}{lll}
l_1 \preceq_{lex} l_2 & \leftrightarrow & (l_1=[] \wedge l_2=[]) \vee \\
 & & (l_1=[x|l_1'] \wedge l_2=[y|l_2'] \wedge x<y) \vee \\
 & & (l_1=[x|l_1'] \wedge l_2=[y|l_2'] \wedge x=y \wedge l_1' \preceq_{lex} l_2')
\end{array}
$$

CHR Implementation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

Lexicographic Order

Definition

Given two sequences l_1 and l_2 of length n, $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$, then $l_1 \preceq_{lex} l_2$ iff $n=0$ or $x_1 < y_1$ or $x_1 = y_1$ and $[x_2, \ldots, x_n] \preceq_{lex} [y_2, \ldots, y_n]$.

Logical Specification

$$
\begin{array}{lll}
l_1 \preceq_{lex} l_2 & \leftrightarrow & (l_1=[] \wedge l_2=[]) \vee \\
& & (l_1=[x|l_1'] \wedge l_2=[y|l_2'] \wedge x<y) \vee \\
& & (l_1=[x|l_1'] \wedge l_2=[y|l_2'] \wedge x=y \wedge l_1' \preceq_{lex} l_2')
\end{array}
$$

CHR Implementation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

Lexicographic Order

Definition

Given two sequences l_1 and l_2 of length n, $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$, then $l_1 \preceq_{lex} l_2$ iff $n=0$ or $x_1 < y_1$ or $x_1 = y_1$ and $[x_2, \ldots, x_n] \preceq_{lex} [y_2, \ldots, y_n]$.

Logical Specification

$$
\begin{array}{lll}
l_1 \preceq_{lex} l_2 & \leftrightarrow & (l_1=[] \wedge l_2=[]) \vee \\
 & & (l_1=[x|l_1'] \wedge l_2=[y|l_2'] \wedge x<y) \vee \\
 & & (l_1=[x|l_1'] \wedge l_2=[y|l_2'] \wedge x=y \wedge l_1' \preceq_{lex} l_2')
\end{array}
$$

CHR Implementation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

First Implemention in CHR

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

Queries

`[1] lex [2]` $\longrightarrow_{l2}$ `true.`

`[X] lex [X]` $\longrightarrow_{l3}$ `[] lex []` $\longrightarrow_{l1}$ `true.`

`[X] lex [Y], X<Y` $\longrightarrow_{l2}$ `X<Y.`

`[X] lex [Y].`

`[X] lex [Y], X>Y.`

`[X] lex [Y], X<>Y.`

No propagation yet :-(

First Implemention in CHR

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

Queries

`[1] lex [2]` $\mapsto_{l2}$ `true.`

`[X] lex [X]` $\mapsto_{l3}$ `[] lex []` $\mapsto_{l1}$ `true.`

`[X] lex [Y], X<Y` $\mapsto_{l2}$ `X<Y.`

`[X] lex [Y].`

`[X] lex [Y], X>Y.`

`[X] lex [Y], X<>Y.`

No propagation yet :-(

First Implemention in CHR

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

Queries

`[1] lex [2]` $\rightarrow_{12}$ `true.`

`[X] lex [X]` $\rightarrow_{13}$ `[] lex []` $\rightarrow_{11}$ `true.`

`[X] lex [Y], X<Y` $\rightarrow_{12}$ `X<Y.`

`[X] lex [Y].`

`[X] lex [Y], X>Y.`

`[X] lex [Y], X<>Y.`

No propagation yet :-(

First Implemention in CHR

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

Queries

[1] lex [2] $\rightarrow_{12}$ true.

[X] lex [X] $\rightarrow_{13}$ [] lex [] $\rightarrow_{11}$ true.

[X] lex [Y], X<Y $\rightarrow_{12}$ X<Y.

[X] lex [Y].

[X] lex [Y], X>Y.

[X] lex [Y], X<>Y.

No propagation yet :-(

First Implemention in CHR

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

Queries

`[1] lex [2]` $\rightarrow_{12}$ `true.`

`[X] lex [X]` $\rightarrow_{13}$ `[] lex []` $\rightarrow_{11}$ `true.`

`[X] lex [Y], X<Y` $\rightarrow_{12}$ `X<Y.`

`[X] lex [Y].`

`[X] lex [Y], X>Y.`

`[X] lex [Y], X<>Y.`

No propagation yet :-(

First Implemention in CHR

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
```

Queries

[1] lex [2] $\rightarrow_{l2}$ true.

[X] lex [X] $\rightarrow_{l3}$ [] lex [] $\rightarrow_{l1}$ true.

[X] lex [Y], X<Y $\rightarrow_{l2}$ X<Y.

[X] lex [Y].

[X] lex [Y], X>Y.

[X] lex [Y], X<>Y.

No propagation yet :-(

Adding Propagation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
```

Queries

[X] lex [Y] $\rightarrow_{l4}$ [X] lex [Y], X=<Y.

[X] lex [Y], X>Y $\rightarrow_{l4}$ [X] lex [Y], X>Y, X=<Y $\rightarrow_{\text{built-in}}$ fail.

[X] lex [Y], X<>Y $\rightarrow_{l4}$ [X] lex [Y], X<Y $\rightarrow_{l2}$ X<Y.

[R1,R2,R3] lex [T1,T2,T3], R2=T2, R3>T3 doesn't add R1<T1 :-(

Adding Propagation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
```

Queries

`[X] lex [Y]` $\rightarrow_{l4}$ `[X] lex [Y], X=<Y.`

`[X] lex [Y], X>Y` $\rightarrow_{l4}$ `[X] lex [Y], X>Y, X=<Y` $\rightarrow_{\text{built-in}}$ `fail.`

`[X] lex [Y], X<>Y` $\rightarrow_{l4}$ `[X] lex [Y], X<Y` $\rightarrow_{l2}$ `X<Y.`

`[R1,R2,R3] lex [T1,T2,T3], R2=T2, R3>T3` doesn't add `R1<T1` :-(

Adding Propagation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
```

Queries

[X] lex [Y] $\rightarrow_{l4}$ [X] lex [Y], X=<Y.

[X] lex [Y], X>Y $\rightarrow_{l4}$ [X] lex [Y], X>Y, X=<Y $\rightarrow_{\text{built-in}}$ fail.

[X] lex [Y], X<>Y $\rightarrow_{l4}$ [X] lex [Y], X<Y $\rightarrow_{l2}$ X<Y.

[R1,R2,R3] lex [T1,T2,T3], R2=T2, R3>T3 doesn't add R1<T1 :-(

Adding Propagation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
```

Queries

[X] lex [Y] $\rightarrow_{l4}$ [X] lex [Y], X=<Y.
[X] lex [Y], X>Y $\rightarrow_{l4}$ [X] lex [Y], X>Y, X=<Y $\rightarrow_{\text{built-in}}$ fail.
[X] lex [Y], X<>Y $\rightarrow_{l4}$ [X] lex [Y], X<Y $\rightarrow_{l2}$ X<Y.

[R1,R2,R3] lex [T1,T2,T3], R2=T2, R3>T3 doesn't add R1<T1 :-(

Adding Simplification

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6 @ [X,U|L1] lex [Y,V|L2] <=> U=V | [X|L1] lex [Y|L2].
```

[R1,R2,R3] lex [T1,T2,T3], R2=T2, R3>T3

$\mapsto_{l6}$ R2=T2, R3>T3, [R1,R3] lex [T1,T3]

$\mapsto_{l5}$ R2=T2, R3>T3, R1<T1.

[R1,R2,R3] lex [T1,T2,T3], R2>=T2, R3>T3 doesn't add R1<T1 :-(

Adding Simplification

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6 @ [X,U|L1] lex [Y,V|L2] <=> U=V | [X|L1] lex [Y|L2].
```

[R1,R2,R3] lex [T1,T2,T3], R2=T2, R3>T3

$\rightarrow_{l6}$ R2=T2, R3>T3, [R1,R3] lex [T1,T3]

$\rightarrow_{l5}$ R2=T2, R3>T3, R1<T1.

[R1,R2,R3] lex [T1,T2,T3], R2>=T2, R3>T3 doesn't add R1<T1 :-(

Adding Simplification

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6 @ [X,U|L1] lex [Y,V|L2] <=> U=V | [X|L1] lex [Y|L2].
```

`[R1,R2,R3] lex [T1,T2,T3], R2=T2, R3>T3`
$\rightarrow_{l6}$ `R2=T2, R3>T3, [R1,R3] lex [T1,T3]`
$\rightarrow_{l5}$ `R2=T2, R3>T3, R1<T1.`

`[R1,R2,R3] lex [T1,T2,T3], R2>=T2, R3>T3` doesn't add `R1<T1` :-(

Turning Simplification into Propagation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6'@ [X,U|L1] lex [Y,V|L2] ==> U>=V | [X|L1] lex [Y|L2].
```

`[R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3` $\rightarrow_{16'}$
`[R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3, [R1,R3]lex[T1,T3]`
$\rightarrow_{15}$ `[R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3, R1<T1`
$\rightarrow_{12}$ `R2>=T2, R3>T3, R1<T1.`

Quadratic worst-case time complexity, if after complete propagation with `16'` $O(n^2)$ variable pairs are removed one by one by rule `13`.

Turning Simplification into Propagation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6'@ [X,U|L1] lex [Y,V|L2] ==> U>=V | [X|L1] lex [Y|L2].
```

[R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3 $\rightarrow_{l6'}$
[R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3, [R1,R3]lex[T1,T3]
$\rightarrow_{l5}$ [R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3, R1<T1
$\rightarrow_{l2}$ R2>=T2, R3>T3, R1<T1.

Quadratic worst-case time complexity, if after complete propagation with l6' $O(n^2)$ variable pairs are removed one by one by rule l3.

Turning Simplification into Propagation

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6'@ [X,U|L1] lex [Y,V|L2] ==> U>=V | [X|L1] lex [Y|L2].
```

[R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3 $\rightarrow_{l6'}$
[R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3, [R1,R3]lex[T1,T3]
$\rightarrow_{l5}$ [R1,R2,R3] lex [T1,T2,T3], R2>=T2,R3>T3, R1<T1
$\rightarrow_{l2}$ R2>=T2, R3>T3, R1<T1.

Quadratic worst-case time complexity, if after complete propagation with l6' $O(n^2)$ variable pairs are removed one by one by rule l3.

Improving Time Complexity

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
```

The culprit propagation rule...

```
l6’ @ [X,U|L1] lex [Y,V|L2] ==> U>=V | [X|L1] lex [Y|L2].
```

Improving Time Complexity

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
```

...is turned into a logically equivalent simplification rule

```
l6' @ [X,U|L1] lex [Y,V|L2] <=> U>=V | [X|L1] lex [Y|L2],
                  [X,U|L1] lex [Y,V|L2].
```

but it does not terminate.

Improving Time Complexity

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
```

...this rule terminates, it is correct and efficient.

```
l6''@ [X,U|L1] lex [Y,V|L2] <=> U>=V,L1=[_|_]|
[X|L1]lex[Y|L2],
                [X,U] lex [Y,V].
```

Worst-Case Time Complexity

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6''@[X,U|L1] lex [Y,V|L2] <=> U>=V, L1=[_ |_ ] |
                                [X,U] lex [Y,V], [X|L1] lex [Y|L2].
```

Measure independent of constraint system (with its own complexity):
Number of atomic built-in constraints that are checked and imposed.
Proportional to the number of rule applications.

Worst-Case Time Complexity

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6''@[X,U|L1] lex [Y,V|L2] <=> U>=V, L1=[_ |_ ] |
                              [X,U] lex [Y,V], [X|L1] lex [Y|L2].
```

Number of rule applications depends on list length n of `lex` constraint.

- Propagation rule `l4` applied at most once.
- Rules `l1`, `l2` and `l5` contribute constant number.
- Recursive rule `l3`: linear in list length.
- Recursive rule `l6''`: linear, as `[X,U] lex [Y,V]` constant number.

Linear in list length n. At most $O(n)$ built-in constraints processed.

Confluence

Result of a query is always the same, no matter which of the applicable rules are applied.

$$
\begin{array}{ccc}
A & \mapsto & B \\
A & \mapsto & C \\
\hline
B & \mapsto^* & D \\
C & \mapsto^* & D
\end{array}
$$

⇒ Independence from the order in which constraints processed.
⇒ Consistency of logical reading of the program.
Decidable, sufficient and necessary condition for confluence of terminating CHR programs through joinability of critical pairs.

Critical pair results from applying two rules to an overlap.
Overlap takes both rule heads and guards, shares some CHR constraints.
Confluence checker available.

Example for `lex`

```
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
```

Critical Pair from Overlap

$$
\begin{tikzcd}
 & {[X,U|L1]\ \texttt{lex}\ [Y,V|L2],\ X=Y,\ U>V} \arrow[dl, "\texttt{l3}"'] \arrow[dr, "\texttt{l5}"] & \\
 {X=Y, U>V, [U|L1]\ \texttt{lex}\ [V|L2]} \arrow[dr, "\texttt{l4}\text{, built-in}"'] & & {X=Y, U>V, X<Y} \arrow[dl, "\text{built-in}"] \\
 & \texttt{fail} &
\end{tikzcd}
$$

Example for `lex`

```
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
```

Critical Pair from Overlap

$$
\begin{tikzcd}
 & {[X,U \mid L1]\ \texttt{lex}\ [Y,V \mid L2],\quad X{=}Y,\quad U{>}V} \arrow[dl, "\texttt{l3}"'] \arrow[dr, "\texttt{l5}"] & \\
{X{=}Y, U{>}V, [U \mid L1]\ \texttt{lex}\ [V \mid L2]} \arrow[dr, "\texttt{l4},\ \text{built-in}"'] & & {X{=}Y, U{>}V, X < Y} \arrow[dl, "\text{built-in}"] \\
 & \texttt{fail} &
\end{tikzcd}
$$

Logical Correctness

```
l1 @ [] lex [] <=> true.
l2 @ [X|L1] lex [Y|L2] <=> X<Y | true.
l3 @ [X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.
l4 @ [X|L1] lex [Y|L2] ==> X=<Y.
l5 @ [X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
l6''@[X,U|L1] lex [Y,V|L2] <=> U>=V, L1=[_|_] | ...
```

Logical Reading

$$
\begin{array}{rlr}
 & ([] \preceq_{lex} []) & \\
X<Y \rightarrow & ([X|L1] \preceq_{lex} [Y|L2]) & \\
X=Y \rightarrow & ([X|L1] \preceq_{lex} [Y|L2] & \leftrightarrow L1 \preceq_{lex} L2) \\
 & ([X|L1] \preceq_{lex} [Y|L2] & \rightarrow X \leq Y) \\
U>V \rightarrow & ([X,U|L1] \preceq_{lex} [Y,V|L2] & \leftrightarrow X < Y) \\
(U \geq V \wedge L1=[\_|\_]) \rightarrow & ([X,U|L1] \preceq_{lex} [Y,V|L2] & \leftrightarrow \\
 & \qquad ([X,U] \preceq_{lex} [Y,V] \wedge [X|L1] & \preceq_{lex} [Y|L2]))
\end{array}
$$

Logical Correctness

Logical Specification

$$
\begin{array}{lll}
l_1 \preceq_{lex} l_2 & \leftrightarrow & (l_1{=}[] \wedge l_2{=}[]) \vee \\
 & & (l_1{=}[x|l_1'] \wedge l_2{=}[y|l_2'] \wedge x{<}y) \vee \\
 & & (l_1{=}[x|l_1'] \wedge l_2{=}[y|l_2'] \wedge x{=}y \wedge l_1' \preceq_{lex} l_2')
\end{array}
$$

Logical Reading must be logical consequence of above specification

$$
\begin{array}{rll}
 & ([] \preceq_{lex} []) & \\
X{<}Y \rightarrow & ([X|L1] \preceq_{lex} [Y|L2]) & \\
X{=}Y \rightarrow & ([X|L1] \preceq_{lex} [Y|L2] & \leftrightarrow L1 \preceq_{lex} L2) \\
 & ([X|L1] \preceq_{lex} [Y|L2] & \rightarrow X{\leq}Y) \\
U{>}V \rightarrow & ([X,U|L1] \preceq_{lex} [Y,V|L2] & \leftrightarrow X < Y) \\
(U{\geq}V \wedge L1{=}[\_|\_]) \rightarrow & ([X,U|L1] \preceq_{lex} [Y,V|L2] & \leftrightarrow \\
 & \quad ([X,U] \preceq_{lex} [Y,V] \wedge [X|L1] & \preceq_{lex} [Y|L2]))
\end{array}
$$

Completeness

Local Consistency: Compute all implied inequalities from `lex` constraint and given inequality.

We already know that the solver is correct and confluent.

Correctness implies: can only propagate too little, not too much / wrong.
Do we propagate enough?

Confluence means: if we find a way to propagate, then we will always propagate, no matter which rules are applied.

Completeness

The `lex` constraint admits $n+1$ solutions of the form

$$x_1{=}y_1 \wedge x_2{=}y_2 \wedge \ldots \wedge x_{i-1}{=}y_{i-1} \wedge x_i{<}y_i \quad (1 \leq i \leq n{+}1),$$

$x_i{<}y_i$ is dropped if $i = n{+}1$.

We write this as $(=)^{i-1}[<]$. i is the position of the solution.

Inequality constraints can be added to a given `lex` constraint so that any subset of solutions is possible.

We want to propagate inequality constraints from the disjunction of these solutions to a given `lex` constraint.

Completeness

- Forward Propagation

From the disjunction we must propagate such that $(=)^{j-1}[\leq]$, where j is the smallest position of any solution.

We can find j by looking for the smallest position that admits $<$.
A position that does not admit a solution must be $>$ or $\geq$.
After $>$ there can be no more solution.
$=$ positions can be ignored.

There is no solution if $(=)^* >$.

$\Rightarrow$ Propagation rule 14 imposes $\leq$ on any current first position and the recursive simplification rule 13 removes leading $=$.

Completeness

- Backward Propagation

If there is only one solution, we must strenghten propagation such that $(=)^{i-1}[<]$.

$\Rightarrow$ Rule `15`: $>$ holds for the second position, so $<$ must hold for the first position. Recursive rule `16''` watches all $\geq$ constraints until $>$.

Conclusions

Lexicographic Order Constraint Solver in CHR

- Executable specification: short, concise
  using recursive decomposition and propagation
- Incremental and concurrent: by nature of CHR
- Independent of underlying constraint system
- Complete: propagates as much as possible
- Efficient: Optimal linear worst-case time complexity
- Confluence: proven by CHR confluence checker
- Correctness: shown by standard CHR analysis

Other approaches: more code, less logical, less analysis.

Conclusions

Lexicographic Order Constraint Solver in CHR

- Executable specification: short, concise
  using recursive decomposition and propagation
- Incremental and concurrent: by nature of CHR
- Independent of underlying constraint system
- Complete: propagates as much as possible
- Efficient: Optimal linear worst-case time complexity
- Confluence: proven by CHR confluence checker
- Correctness: shown by standard CHR analysis

Future work:

- Combine with other CHR solvers
- Performance analysis, benchmarking
- Extensions: `lex` chains, with summation, symmetry breaking
- Automatic derivation from specification possible?