CHR Grammars with Multiple constraint stores

Henning Christiansen

Roskilde University, Computer Science,
P.O.Box 260, DK-4000 Roskilde, Denmark
henning@ruc.dk, http://www.ruc.dk/~henning

Overview

– Introduction to CHR Grammars
– Abductive language interpretation in CHRG
– The need for multiple constraint stores to handle all interp’s in parallel
– Proposal for new architecture optimized for multiple constraint stores
– Conclusion
Introduction to CHR Grammars

Prolog \longrightarrow \text{CHR}

↓ \quad ↓

DCG \longrightarrow \text{CHR Grammars}

Example: Grammar for simple sentences with coordination

“Peter likes and Mary detests spinach”

\[\text{subj}(A), \text{verb}(V), \text{obj}(B) \ Algorithm \ \text{sent}(s(A,V,B)).\]
\[\text{subj}(A), \text{verb}(V) /- [\text{and}], \text{sent}(s(_,_,B)) \ Algorithm \ \text{sent}(s(A,V,B)).\]

Notice context sensitive rules: /- indicates right context.

Compiles into

\[\text{subj}(N0,N1,A), \text{verb}(N1,N2,V), \text{obj}(N2,N3,B) \Rightarrow \text{sent}(N0,N3,s(A,V,B)).\]
\[\text{subj}(N0,N1,A), \text{verb}(N1,N2,V), \text{token}(N2,N3,\text{and}),\]
\[\text{sent}(N3,N4,s(_,_,B)) \Rightarrow \text{sent}(N0,N2,s(A,V,B)).\]
General properties of CHRG

- Syntactic sugar over CHR.
- High expressibility: contains full power of CHR (and Prolog) plus ...
- Several advantages for NLP; documented elsewhere
- Implemented by compile-on-load → CHR
  ... followed by CHR → Prolog → ...
- Fully equipped with top-level pred’s, auxiliaries, passive optimizations
- Available for SICSTUS Prolog on www
Abductive language interpretation in CHRG

Basically following scheme of [Abdennadher, Christiansen, FQAS200]:

- Abducibles $\rightarrow$ CHR constraints
- Integrity constraints $\rightarrow$ CHR rules

Efficient: No metainterpreter
Price: Slightly limited use of negation compared with some abd. frameworks

Translating abductive language interpretation into deduction...
Translating abductive language interpretation into deduction

Rule perfect for deductive verification of correct utterance wrt. known semantic context:

[help], {have_problem} ::= exclamation.

Move “abducibles” to other side of implication:

A rule perfect for abductive reasoning to figure out feasible semantic context

[help] ::= exclamation, {have_problem}.

Easy to formalize and prove correct [Christiansen, NLULP2002, TPLP2005]

Works perfectly for unambiguous grammars

In general, extra machinery needed to avoid mixing up different interpretations
Problem with abduction and ambiguous grammars

Delivers all grammar nodes (syntax trees) and their abducibles in one big mess
Solution: Each syntax tree has its own local constraint store

Example: Three syntax trees each with its set of abducibles in local store

$\Delta_1^C$, $\Delta_2^C$, $\Delta_3^C$

Assume rules so $\Delta_1 \Rightarrow \Delta_{12}, \{c_{12}\}$ and $\Delta_2 \Rightarrow \Delta_{23}, \{c_{23}\}$

Then derivation should produce new trees

$\Delta_{12}^{C_1 \cup C_2 \cup C_{12}}$, $\Delta_{23}^{C_2 \cup C_3 \cup C_{23}}$

and apply integrity constraints $\approx$ CHR rules to the new local stores

Can be implemented in SICStus version of CHR with indexing technique and copying of constraints

Needless to say: Very inefficient
Proposal for a new and optimized architecture
written in C

- Goal: Evaluate all different abductive interpretations in parallel
- efficient copying of constraints (therefore C)
- with efficient pruning by integrity constraints

Sketch of a prototype currently under development

- Several details still need to be worked out – suggestions, comments well-come
- Ignored in this talk: Context sensitive rules implies restrictions on when
  grammar rule can apply
Procedural semantics

A multi constraint store: a set of pairs \((G, S)\) where \(G\) a grammar symbol and \(S\) a constraint store of abducible atoms.

A derivation is a sequence of m. c. stores \(M_1, \ldots, M_n\), such that:

- \(M_1\) has one component for each input token, e.g.:
  \[M_1 = \{(\text{token}(0,1,\text{the}), \emptyset), (\text{token}(1,2,\text{man}), \emptyset), (\text{token}(2,3, \text{walks}), \emptyset)\}\]

- \(M_{i+1}\) is derived from \(M_i\) if either:
  - There is an instance of a grammar rule \(G_1, \ldots, G_m : \rightarrow G_0, \{A\}\) with \(\langle G_j, S_j \rangle \in M_i\) for \(j = 1 \ldots m\); \(A\) a set of abducibles. Then
    \[M_{i+1} = M_i \cup \{\langle G_0, S_1 \cup \cdots \cup S_m \cup A \rangle\}\]

  - An integrity constraint can apply to a local constraint store \(S\) in \(M_i\) producing a new store \(S'\) and possible sideeffects on variables in \(G\) given by substitution \(\sigma\). Then
    \[M_{i+1} = M_i \setminus \{\langle G, S \rangle\} \cup \{\langle G \sigma, S' \rangle\}\]
    However, if \(S'\) is failed, \(M_{i+1} = M_i \setminus \{\langle G, S \rangle\}\).

- No rule is applied twice to the same constraints.

Detailed computation rule ...
Detailed computation rule:

- Components of the $M_0$ are entered one by one from left to right, each at a point when no rule can apply.
- Whenever a grammar rule is applied, integrity constraints apply as long as possible before a grammar rule can apply.
Representation local constraint stores

Relative addressing so that copying becomes easy.
Example with unary abducibles \( p(X), q(Y), r(Z) \)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>variable offset=0</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>variable offset=0</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>variable offset=0</td>
<td></td>
</tr>
</tbody>
</table>

Unify \( X=Y \):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>variable offset=1</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>variable offset=0</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>variable offset=0</td>
<td></td>
</tr>
</tbody>
</table>

Unify \( X=a, Z=Y \):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>variable offset=1</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>constant=a</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>variable offset=-1</td>
<td></td>
</tr>
</tbody>
</table>

Observe: Combining diff. stores \( \approx \) copy into consecutive cells
Deletion caused by propagation rule: marking bit
Ideas for optimization based on techniques from data integration

[Christiansen, Martinenghi, LOPSTR2004, FOIKS2004]

Assume $\Gamma(C)$ is the set of ICs testing a set local constraint store

- Let $\Gamma'(C_1, C_2)$ be a specialized and optimized IC based on assumptions
  - $C_1$ and $C_2$ are consistent separately, i.e., all rules have applied
- Let $\Gamma''(C_1, A)$ be a specialized and optimized IC based on assumptions
  - $C_1$ is consistent, i.e., all rules have applied
  - $A$ is known in advance (apart from instantiation of variables)

Incremental algorithm for copying and integrity checking of $S_{\text{new}} = S_1 \cup \cdots \cup S_m \cup A$:

1. $i := 1$; $S_{\text{new}} := S_1$
2. $S_{\text{new}} := \Gamma'(S_{\text{new}}, S_i)$
3. if $S_{\text{new}} = \text{failure}$ then failure-exit;
4. if $i < m$ then \{ $i := i + 1$; go to 2 \};
5. $S_{\text{new}} := \Gamma''(S_{\text{new}}, A)$
6. if $S_{\text{new}} = \text{failure}$ then failure-exit else normal-exit;

((Ignored addition of new abducibles by ICs; easy to add))
Conclusion

– Ongoing implementation; no disappointments yet
– Ambition: To produce a barrier-breaking environment for abductive language interpretation
  • Flexible language for expressing detailed and strong ICs
  • ... needed to keep down explosive complexity
  • Very efficient check and pruning
  • Pointing forward to language processing with all phases in cooperative parallel: Lexical, morphologic, ontologic, semantics, pragmatic ...

:)