Recognizing Strategies

Bastiaan Heeren [1] Johan Jeuring [1,2]

[1] Open Universiteit Nederland
[2] Universiteit Utrecht, The Netherlands

14 July 2008 (WRS'08)
Castle of Hagenberg, Austria

Overview

Introduction to exercise assistants

Strategies for exercises

A strategy recognizer

Conclusions

Introduction to exercise assistants

- A rewrite system for logical propositions:

$$\neg\neg p \Rightarrow p \qquad p \wedge (q \vee r) \Rightarrow (p \wedge q) \vee (p \wedge r)$$

$$\neg(p \wedge q) \Rightarrow \neg p \vee \neg q \qquad (p \vee q) \wedge r \Rightarrow (p \wedge r) \vee (q \wedge r)$$

$$\neg(p \vee q) \Rightarrow \neg p \wedge \neg q$$

- Exercise: bring proposition to disjunctive normal form

$$\neg(\neg(p \vee q) \wedge r)$$

Introduction to exercise assistants

- A rewrite system for logical propositions:

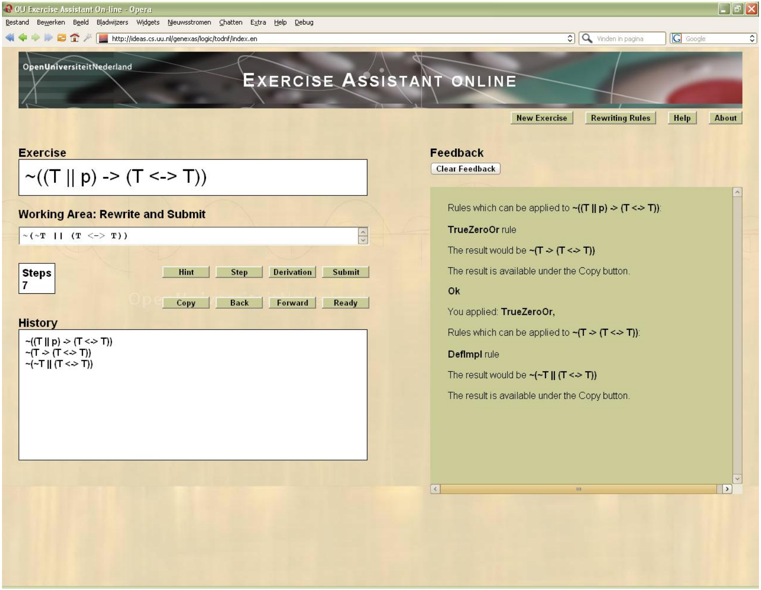
$$\neg\neg p \Rightarrow p \qquad p \wedge (q \vee r) \Rightarrow (p \wedge q) \vee (p \wedge r)$$
$$\neg(p \wedge q) \Rightarrow \neg p \vee \neg q \qquad (p \vee q) \wedge r \Rightarrow (p \wedge r) \vee (q \wedge r)$$
$$\neg(p \vee q) \Rightarrow \neg p \wedge \neg q$$

- Exercise: bring proposition to disjunctive normal form

$$\begin{array}{ll} & \neg(\neg(p \vee q) \wedge r) \\ \Rightarrow & \neg\neg(p \vee q) \vee \neg r \\ \Rightarrow & p \vee q \vee \neg r \end{array}$$

- Exercise is solved in just two steps

Introduction to exercise assistants (2)



Introduction to exercise assistants (3)

- A different derivation (same proposition):

$$
\begin{array}{ll}
 & \neg(\neg(p \vee q) \wedge r) \\
\Rightarrow & \neg((\neg p \wedge \neg q) \wedge r) \\
\Rightarrow & \neg(\neg p \wedge \neg q) \vee \neg r \\
\Rightarrow & \neg\neg p \vee \neg\neg q \vee \neg r \\
\Rightarrow & p \vee \neg\neg q \vee \neg r \\
\Rightarrow & p \vee q \vee \neg r
\end{array}
$$

- Same answer, more steps

Introduction to exercise assistants (3)

- A different derivation (same proposition):

$$\begin{array}{ll} & \neg(\neg(p \vee q) \wedge r) \\ \Rightarrow & \neg((\neg p \wedge \neg q) \wedge r) \\ \Rightarrow & \neg(\neg p \wedge \neg q) \vee \neg r \\ \Rightarrow & \neg\neg p \vee \neg\neg q \vee \neg r \\ \Rightarrow & p \vee \neg\neg q \vee \neg r \\ \Rightarrow & p \vee q \vee \neg r \end{array}$$

- Same answer, more steps

Expert strategy for DNF exercise:

- First push negations inside (top-down)
- Then use the distribution rule

Strategies for exercises

We have defined a strategy language for exercises with:

1. Transformation rules	
2. Sequence	$s \lt\!\star\!\gt t$
3. Choice	$s \lt\!|\!\gt t$
4. Unit elements	*succeed*, *fail*
5. Labels	*label* ℓ *s*
6. Recursion	*fix* *f*

- Labels are used to mark positions in a strategy
- Combinators are inspired by context-free grammars
- In fact, this is an embedded domain specific language (in Haskell) and more combinators can be added:

$$\textit{many}\ s = \textit{fix}\ (\lambda x \rightarrow \textit{succeed} \lt\!|\!\gt (s \lt\!\star\!\gt x))$$

Strategies for exercises (2)

- A strategy specification for the DNF exercise:

$$negations = deMorganAnd <|> deMorganOr <|> doubleNeg$$

$$\begin{aligned} dnf = label\ \ell_0\ \ (\ & label\ \ell_1\ (repeat\ (topDown\ negations)) \\ <\star> & label\ \ell_2\ (repeat\ (somewhere\ andOverOr)) \\ )\ & \end{aligned}$$

- The strategy contains four rewrite rules
- *repeat* is a greedy variation of the *many* combinator
- *topDown* and *somewhere* are traversal combinators

Strategy recognition and grammars

- t_0 initial term (or exercise)
- $t_1, t_2, \ldots$ terms submitted by the student
- $r_0, r_1, \ldots$ rules recognized by the system

$$t_0 \xrightarrow{r_0} t_1 \xrightarrow{r_1} t_2 \xrightarrow{r_2} t_3 \xrightarrow{r_3} \ldots$$

Strategy recognition: Is the sequence of rules “valid” according to the strategy?

Strategy recognition and grammars

- t_0 initial term (or exercise)
- $t_1, t_2, \ldots$ terms submitted by the student
- $r_0, r_1, \ldots$ rules recognized by the system

$$t_0 \xrightarrow{r_0} t_1 \xrightarrow{r_1} t_2 \xrightarrow{r_2} t_3 \xrightarrow{r_3} \ldots$$

Strategy recognition: Is the sequence of rules "valid" according to the strategy?

Key observation: tracking intermediate rewrite steps is a parsing problem:

> "Is the sequence of rules a prefix of a sentence in the language generated by the strategy?"

A strategy recognizer

- Our paper discusses the design and implementation of a strategy recognizer
- Why not reuse an existing parser library?
  1. Only interested in sequences of rules that can be applied successively to some initial term
  2. Also prefixes have to be recognized
  3. Error diagnosis is important for high-quality feedback
  4. Recognizer must deal with labels
  5. Strategy should be serializable

Representing grammars

- A data type for grammars:

$$
\begin{aligned}
\mathbf{data}\ \mathit{Grammar}\ a = {} & \mathit{Grammar}\ a :\star: \mathit{Grammar}\ a \\
\mid {} & \mathit{Grammar}\ a :|: \mathit{Grammar}\ a \\
\mid {} & \mathit{Rec}\ \mathit{Int}\ (\mathit{Grammar}\ a) \\
\mid {} & \mathit{Symbol}\ a \mid \mathit{Var}\ \mathit{Int} \mid \mathit{Succeed} \mid \mathit{Fail}
\end{aligned}
$$

Representing grammars

- A data type for grammars:

$$\begin{aligned}\textbf{data}\ \mathit{Grammar}\ a = {} & \mathit{Grammar}\ a :\star: \mathit{Grammar}\ a \\ \mid {} & \mathit{Grammar}\ a :|: \mathit{Grammar}\ a \\ \mid {} & \mathit{Rec}\ \mathit{Int}\ (\mathit{Grammar}\ a) \\ \mid {} & \mathit{Symbol}\ a \mid \mathit{Var}\ \mathit{Int} \mid \mathit{Succeed} \mid \mathit{Fail}\end{aligned}$$

- Smart constructors for simplification:

$$\begin{aligned}&(<|>) :: \mathit{Grammar}\ a \rightarrow \mathit{Grammar}\ a \rightarrow \mathit{Grammar}\ a \\ &\mathit{Fail}\ <|>\ t = t \\ &s\ <|>\ \mathit{Fail} = s \\ &(s :|: t)\ <|>\ u = s :|: (t <|> u) \\ &s\ <|>\ t = s :|: t\end{aligned}$$

The function *empty*

Is the empty sequence in the language?

$$
\begin{aligned}
&\mathit{empty} :: \mathit{Grammar}\ a \rightarrow \mathit{Bool}\\
&\mathit{empty}\ (s :\star: t) &&= \mathit{empty}\ s \wedge \mathit{empty}\ t\\
&\mathit{empty}\ (s :|: t) &&= \mathit{empty}\ s \vee \mathit{empty}\ t\\
&\mathit{empty}\ (\mathit{Rec}\ i\ s) &&= \mathit{empty}\ s\\
&\mathit{empty}\ \mathit{Succeed} &&= \mathit{True}\\
&\mathit{empty}\ \_ &&= \mathit{False}
\end{aligned}
$$

- There is no need for *empty* to inspect recursive occurrences of a grammar
- Straightforward definition for the other cases

The function *firsts*

Which symbols can appear first in a sentence, and what is the remaining grammar?

$$
\begin{aligned}
&\mathit{firsts} :: \mathit{Grammar}\ a \rightarrow [(a, \mathit{Grammar}\ a)] \\
&\mathit{firsts}\ (s :\!\star\!: t) &&= [(a, s' \lessdot\!\star\!\gtrdot t) \mid (a, s') \leftarrow \mathit{firsts}\ s] \mathbin{+\!\!+} \\
& &&\quad (\mathbf{if}\ \mathit{empty}\ s\ \mathbf{then}\ \mathit{firsts}\ t\ \mathbf{else}\ [\,]) \\
&\mathit{firsts}\ (s :|: t) &&= \mathit{firsts}\ s \mathbin{+\!\!+} \mathit{firsts}\ t \\
&\mathit{firsts}\ (\mathit{Rec}\ i\ s) &&= \mathit{firsts}\ (\mathit{replaceVar}\ i\ (\mathit{Rec}\ i\ s)\ s) \\
&\mathit{firsts}\ (\mathit{Symbol}\ a) &&= [(a, \mathit{succeed})] \\
&\mathit{firsts}\ \_ &&= [\,]
\end{aligned}
$$

- We unfold a recursive grammar with *replaceVar*
- With *empty* and *firsts* we can run a strategy, and trace submitted steps

Running a strategy

$$
\begin{aligned}
&\mathit{run} :: \mathit{Grammar}\ (\mathit{Rule}\ a) \rightarrow a \rightarrow [a] \\
&\mathit{run}\ s\ a = [a \mid \mathit{empty}\ s] \\
&\qquad\quad +\!\!+\ [c \mid (r, t) \leftarrow \mathit{firsts}\ s, b \leftarrow \mathit{apply}\ r\ a, c \leftarrow \mathit{run}\ t\ b]
\end{aligned}
$$

- Results are returned in a depth-first manner
- What about labels in the strategy?

Labeled strategies

Labels are excluded from the *Grammar* data type, which makes it simpler to manipulate grammars

- Two mutually recursive types:

$$\textbf{data}\ \mathit{LabStrat}\ \ell\ a = \mathit{Label}\ \ell\ (\mathit{Strategy}\ \ell\ a)$$

$$\textbf{type}\ \mathit{Strategy}\ \ell\ a = \mathit{Grammar}\ (\mathit{Either}\ (\mathit{Rule}\ a)\ (\mathit{LabStrat}\ \ell\ a))$$

- Rules are tagged *Left*, nested labels are tagged *Right*
- For convenience, all smart constructors are overloaded to circumvent tagging

Labeled strategies (2)

- We can now trace where we are in the strategy:

$$\textbf{data}\ \mathit{Step}\ \ell\ a = \mathit{Enter}\ \ell \mid \mathit{Step}\ (\mathit{Rule}\ a) \mid \mathit{Exit}\ \ell$$

$$\begin{array}{l} \mathit{withSteps} :: \mathit{LabStrat}\ \ell\ a \rightarrow \mathit{Grammar}\ (\mathit{Step}\ \ell\ a) \\ \mathit{withSteps}\ (\mathit{Label}\ \ell\ s) \ = \ \mathit{symbol}\ (\mathit{Enter}\ \ell) \\ \qquad\qquad\qquad\qquad\quad <\!\star\!>\ \mathit{mapSymbol}\ f\ s \\ \qquad\qquad\qquad\qquad\quad <\!\star\!>\ \mathit{symbol}\ (\mathit{Exit}\ \ell) \\ \quad \textbf{where} \\ \qquad f = \mathit{either}\ (\mathit{symbol} \circ \mathit{Step})\ \mathit{withSteps} \end{array}$$

- *Enter* ℓ and *Exit* ℓ are administrative steps
- Some strategy combinators introduce administrative rules

Tracing with labels: an example

$$
\begin{aligned}
\mathit{dnf} = \mathit{label}\ \ell_0\ \ & (\ \ \mathit{label}\ \ell_1\ (\mathit{repeat}\ (\mathit{topDown}\ \mathit{negations})) \\
& \lt\!\star\!\gt \mathit{label}\ \ell_2\ (\mathit{repeat}\ (\mathit{somewhere}\ \mathit{andOverOr})) \\
&)
\end{aligned}
$$

- Running *dnf* on $\neg(\neg(p \vee q) \wedge r)$ with steps returns:

$$
\begin{aligned}
[\,&\mathit{Enter}\ \ell_0, \mathit{Enter}\ \ell_1, \mathit{Step}\ \mathit{deMorganAnd}, \\
&\quad \mathit{Step}\ \mathit{not}, \mathit{Step}\ \mathit{down}, \mathit{Step}\ \mathit{doubleNeg}, \mathit{Step}\ \mathit{up}, \\
&\quad \mathit{Step}\ \mathit{not}, \\
&\quad \mathit{Exit}\ \ell_1, \\
&\quad \mathit{Enter}\ \ell_2, \mathit{Step}\ \mathit{not}, \\
&\quad \mathit{Exit}\ \ell_2, \\
&\mathit{Exit}\ \ell_0\,]
\end{aligned}
$$

Extensions

Our paper discusses some extensions:

1. Parallel strategies
   - Without the problems usually encountered
2. Removing left recursion
   - Because our grammars can be inspected
3. Serializing the remaining strategy
   - For establishing a binding with other e-learning environments

These extensions illustrate the flexibility of our approach

Conclusions

- The paper presents the design and implementation of a strategy recognizer
- Tracking student steps can be viewed as a parsing problem
- Experience in parsing context-free languages can be transferred to exercise assistants
- Our grammar representation is observable, also during parsing, which helps in diagnosing errors

We are very much interested in learning more about "recognizing strategies" in different areas