The DynGenPar Algorithm on an Example

and a comparison with existing approaches

Kevin Kofler and Arnold Neumaier

University of Vienna, Faculty of Mathematics

July 16, 2011

Acknowledgements

Support by the Austrian Science Fund (FWF) under contract number P20631 is gratefully acknowledged.

Example Grammar

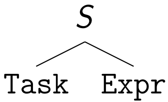
- simple grammar for unconstrained polynomial optimization problems
- $N = \{S, \texttt{Task}, \texttt{Expr}, \texttt{Term}, \texttt{Factor}\}$
- $T = \{\min, \max, +, *, x, \texttt{NUMBER}\}$
- $S \to \texttt{Task Expr}$
- $\texttt{Task} \to \min \mid \max$
- $\texttt{Expr} \to \texttt{Expr} + \texttt{Term} \mid \texttt{Term}$
- $\texttt{Term} \to \texttt{Term} * \texttt{Factor} \mid \texttt{Factor}$
- $\texttt{Factor} \to x \mid \texttt{NUMBER}$
- Example sentence: $\min x * x$

Example Top-Down Parsing (1/3)

- input: $. \min x * x$ (dot ... cursor position)
- current sentence form: $. \ S$
- current parse tree: S
- apply $S \rightarrow \texttt{Task Expr}$

Example Top-Down Parsing (2/3)

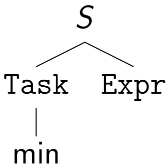
- input: $.\min x * x$ (dot . . . cursor position)
- current sentence form: . `Task Expr`
- current parse tree:



- possible choices:
  1. `Task` $\longrightarrow$ min
  2. `Task` $\longrightarrow$ max
- only rule 1 matches input token min
- apply `Task` $\longrightarrow$ min

Example Top-Down Parsing (3/3)

- min is already a terminal, accept it
- input: min . $x * x$ (dot . . . cursor position)
- current sentence form: min . Expr
- current parse tree:



- possible choices:
  1. Expr $\rightarrow$ Expr $+$ Term
  2. Expr $\rightarrow$ Term
- Problem: Which rule to apply? (Left recursion!)

Example Bottom-Up Parsing (1/5)

- current sentence form: $.\,\mathrm{min}\, x * x$ (dot . . . cursor position)
- no input read in yet
  - thus cannot reduce anything
- only option: shift a token

Example Bottom-Up Parsing (2/5)

- current sentence form: min . $x * x$
- left of the cursor: min
  - right hand side of `Task` $\rightarrow$ min
- no further shift makes sense
  - $\nexists$ rule $X \rightarrow$ min . . .
- thus reduce `Task` $\rightarrow$ min

Example Bottom-Up Parsing (3/5)

- current sentence form: `Task` . $x * x$
- left of the cursor: `Task`
  - not the right hand side of a rule
  - start of $S \longrightarrow$ `Task Expr`
  - thus need more tokens
- perform a shift step

Example Bottom-Up Parsing (4/5)

- current sentence form: `Task` x . $*$ x
- reduce step: `Factor` $\longrightarrow x$
- current sentence form: `Task Factor` . $*$ x
- reduce step: `Term` $\longrightarrow$ `Factor`
- current sentence form: `Task Term` . $*$ x
- what not?
  - reduce `Expr` $\longrightarrow$ `Term`?
  - or shift step?
  - for that, consider the token after the cursor
    - 1 Token *Lookahead* $\rightsquigarrow$ *LR(**1**) method*
  - with lookahead $*$, only a shift makes sense

Example Bottom-Up Parsing (5/5)

- current sentence form: Task Term $*$. x
- shift step
- current sentence form: Task Term $*$ x .
- reduce step: Factor $\rightarrow$ x
- current sentence form: Task Term $*$ Factor .
- reduce step: Term $\rightarrow$ Term $*$ Factor
- current sentence form: Task Term .
- lookahead empty, no more shift possible
  - reduce step: Expr $\rightarrow$ Term
- current sentence form: Task Expr .
- lookahead empty, no more shift possible
  - reduce step: $S \rightarrow$ Task Expr
- we obtain the sentence form: S . $\Rightarrow$ done!

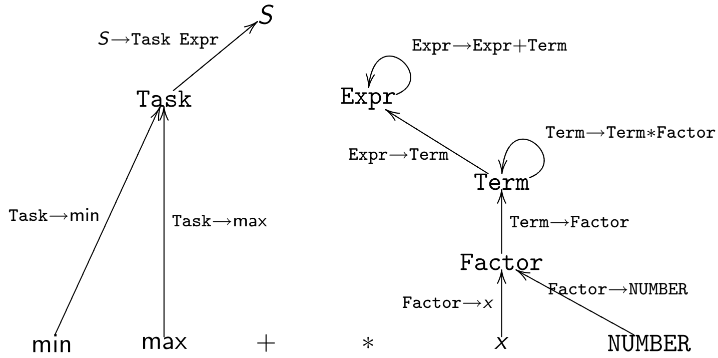
Initial Graph

- replaces precompiled tables
  - dynamically extensible for new rules
- directed, labeled multigraph on $\Gamma = N \cup T$
- tokens T are sources
- edge from symbol $s \in \Gamma$ to category $n \in N$ iff
  - $\exists$ rule $n \to n_1 \; n_2 \; \ldots \; n_k \; s \; \ldots$ with $n_i \in N_0 \; \forall i$
    - where $N_0 \subseteq N$ the set of all nonterminals from which ε can be derived
  - label of the edge
    - that rule
    - number k of n_i set to ε
  - more than one possible label … multi-edge

Example Initial Graph (1/2)

- Example grammar
  - $N = \{S, \texttt{Task}, \texttt{Expr}, \texttt{Term}, \texttt{Factor}\}$
  - $T = \{\text{min}, \text{max}, +, *, x, \texttt{NUMBER}\}$
  - $S \to \texttt{Task Expr}$
  - $\texttt{Task} \to \text{min} \mid \text{max}$
  - $\texttt{Expr} \to \texttt{Expr} + \texttt{Term} \mid \texttt{Term}$
  - $\texttt{Term} \to \texttt{Term} * \texttt{Factor} \mid \texttt{Factor}$
  - $\texttt{Factor} \to x \mid \texttt{NUMBER}$
- $N_0 = \emptyset$
  - because there is no rule $n \to \varepsilon$ in the grammar
  - thus consider only rules of the form $n \to s \ \ldots$
  - k (number of skipped ε categories) $= 0$ everywhere

Example Initial Graph (2/2)

- we obtain the graph
  - 

Neighborhoods

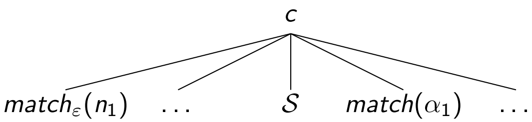
- Let $s \in \Gamma = N \cup T$ (*symbol*), $z \in N$ (*target*)
- *Neighborhood* $\mathcal{N}(s, z)$ ...
  - Edges from s to a category c where
    - z reachable from c
- in the example
  - $\mathcal{N}(\mathrm{min}, S) = \{\mathtt{Task} \rightarrow \mathrm{min}\}$
  - $\mathcal{N}(x, S) = \emptyset$
  - $\mathcal{N}(x, \mathtt{Expr}) = \{\mathtt{Factor} \rightarrow x\}$
  - $\mathcal{N}(\mathtt{Term}, \mathtt{Expr}) = \{\mathtt{Expr} \rightarrow \mathtt{Term}, \mathtt{Term} \rightarrow \mathtt{Term} * \mathtt{Factor}\}$
- computed by graph walk
- can be cached
  - but must be recomputed if the grammar changes

Operations

- $match_\varepsilon(n), n \in N_0$ ... derive ε from n
  - top-down
  - ignore recursion (would produce ∞ly many parse trees)
- *shift* ... read in the next token
- $reduce(s, z), s \in \Gamma, z \in N$... reduce s to z
  - different from LR *reduce*!
  - already reduce after the first symbol
  - *reduce* must complete the match
- $match(s), s \in \Gamma = N \cup T$
  - if $s \in N_0$: $match_\varepsilon(s)$, remember result
  - $t = shift$
  - if $s \in T$: compare s with t
  - if $s \in N$: $reduce(t, s)$

Reduce(s,z) Step

- pick a rule $c \to n_1 \; n_2 \; \ldots \; n_k \; s \; \alpha_1 \; \alpha_2 \; \ldots \; \alpha_\ell$ in $\mathcal{N}(s, z)$
- for each $n_i \in N_0$:
  - $match_\varepsilon(n_i)$
- s already recognized . . . parse tree $\mathcal{S}$
- for each $\alpha_j \in \Gamma = N \cup T$:
  - $match(\alpha_j)$ (top-down step)
- parse tree:



- if $c \neq z$: continue reducing recursively
  - $reduce(c, z)$
- (also consider $reduce(z, z)$ for left recursion)

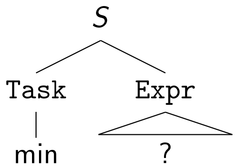
Example DynGenPar Algorithm (1/9)

- in the example: $N_0 = \emptyset \Rightarrow$ no $match_\varepsilon$ steps
- input: $.\,\mathrm{min}\, x * x$ (dot ... cursor position)
- begin with $match(S)$ (start category)
  - shift step produces min
  - next step: $reduce(\mathrm{min}, S)$
  - input now: $\mathrm{min}\,.\; x * x$

Example DynGenPar Algorithm (2/9)

- $reduce(\text{min}, S)$
  - $\mathcal{N}(\text{min}, S) = \{\texttt{Task} \rightarrow \text{min}\}$
  - thus reduce $\texttt{Task} \rightarrow \text{min}$
  - $k = 0$ (no n_i), $\ell = 0$ (no α_j)
  - thus continue with $reduce(\texttt{Task}, S)$
    - $\mathcal{N}(\texttt{Task}, S) = \{S \rightarrow \texttt{Task Expr}\}$
    - thus reduce $S \rightarrow \texttt{Task Expr}$
    - now we have an α_j: $\alpha_1 = \texttt{Expr}$
    - thus $match(\texttt{Expr})$
    - then reduce complete, because S is already the goal

Example DynGenPar Algorithm (3/9)

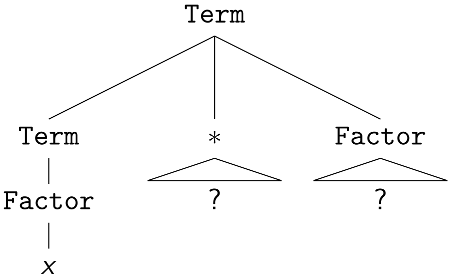
- already recognized:
  - 

- *match*(Expr)
  - shift step produces x
  - next step: $reduce(x, \texttt{Expr})$
  - input now: min x $. * x$

Example DynGenPar Algorithm (4/9)

- *reduce*$(x, \texttt{Expr})$
  - $\mathcal{N}(x, \texttt{Expr}) = \{\texttt{Factor} \to x\}$
  - continue with *reduce*$(\texttt{Factor}, \texttt{Expr})$
  - $\mathcal{N}(\texttt{Factor}, \texttt{Expr}) = \{\texttt{Term} \to \texttt{Factor}\}$
  - continue with *reduce*$(\texttt{Term}, \texttt{Expr})$
  - $\mathcal{N}(\texttt{Term}, \texttt{Expr}) = \{\texttt{Expr} \to \texttt{Term}, \texttt{Term} \to \texttt{Term} * \texttt{Factor}\}$
    - reduce-reduce conflict
    - must consider both possibilities
    - $\texttt{Expr} \to \texttt{Term}$: *reduce*$(x, \texttt{Expr})$ and thus *match*$(\texttt{Expr})$ terminates (or try reducing $\texttt{Expr} \to \texttt{Expr}+\texttt{Term} \Rightarrow$ error), thus also *reduce*(min, S) and *match*(S), but the input is not consumed yet $\Rightarrow$ error
    - thus reduce $\texttt{Term} \to \texttt{Term} * \texttt{Factor}$
    - thus *match*$(*)$ and *match*$(\texttt{Factor})$

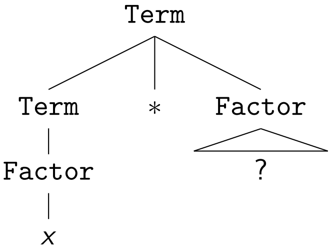
Example DynGenPar Algorithm (5/9)

- already recognized:
  - 

- *match*$(*)$ trivial (only shift)
- input now: $\min x * . \; x$

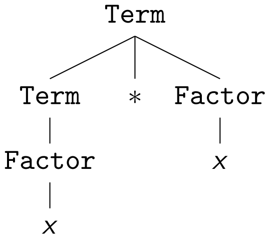
universität wien

Example DynGenPar Algorithm (6/9)

- already recognized:
  - 

- *match*(Factor)
  - shift step produces x
  - next step: *reduce*(x, Factor)
  - input now consumed: min $x * x$.

Example DynGenPar Algorithm (7/9)

- $\textit{reduce}(x, \texttt{Factor})$
  - $\mathcal{N}(x, \texttt{Factor}) = \{\texttt{Factor} \to x\}$
  - so we are already done
- $\texttt{Term}$ recognized completely
  - 

- remaining open target: *Expr*
  - thus continue with another $\textit{reduce}(\texttt{Term}, \texttt{Expr})$

Example DynGenPar Algorithm (8/9)

- *reduce*$(\texttt{Term}, \texttt{Expr})$
  - $\mathcal{N}(\texttt{Term}, \texttt{Expr}) = \{\texttt{Expr} \to \texttt{Term}, \texttt{Term} \to \texttt{Term} * \texttt{Factor}\}$
    - again 2 possibilities
    - this time, $\texttt{Term} \to \texttt{Term}{*}\texttt{Factor}$ fails (out of input)
    - thus reduce $\texttt{Expr} \to \texttt{Term}$
    - continue with *reduce*$(\texttt{Expr}, \texttt{Expr})$
    - $\texttt{Expr} \to \texttt{Expr}{+}\texttt{Term}$ fails (out of input)
    - thus we are done
- and so the input has been recognized completely

Example DynGenPar Algorithm (9/9)

- we obtain the same parse tree as for LR:
  -