1.

Lösungen der Teilaufgaben wie folgt (Ich brauchte nur eine Zeile wegen des Abstands):

(a)	(b)	(c)	(d)	(e)	(f)
Wahr	Falsch	Falsch	Wahr	Wahr	Wahr

2.

Zunächst legen wir die Attribute der Regeln fest, hierzu benötigen wir bei `cmd_seq` und `command` je einen Zähler für `if`-Anweisungen bzw. Zuweisungen. Außerdem benötigen wir für die Knoten den jeweiligen maximalen sowie minimalen Wert, außerdem müssen wir dort aufpassen, dass uns evtl. auch eine boolesche Konstante begegnen kann, hierfür das zusätzliche Symbol `NaN`. `lexval` sei ein vom Lexer bestimmter Wert, da die Grammatik hierauf nicht ausführlicher eingeht.

```
program    -
block      -
cmd_seq    ifcount, assigncount: ℕ0; minval, maxval: ℤ ∪ {NaN}
command    ifcount, assigncount: ℕ0; minval, maxval: ℤ ∪ {NaN}
expression minval, maxval: ℤ ∪ {NaN}
term       minval, maxval: ℤ ∪ {NaN}
factor     minval, maxval: ℤ ∪ {NaN}
"number"   lexval: ℤ
```

Die semantischen Regeln wären dann folgende:

```
block      = declarationlist "begin" cmd_seq "end".
       cond $3.assigncount > $3.ifcount
command    = "identifier" ":=" expression.
       $$.assigncount := 1
       $$.ifcount := 0
       $$.minval := $3.minval
       $$.maxval := $3.maxval
command    = skip.
       $$.assigncount := 0
       $$.ifcount := 0
       $$.minval := NaN
       $$.maxval := NaN
command    = "while" expression "do" cmd_seq "end".
       $$.assigncount := $4.assigncount
       $$.ifcount := $4.ifcount
       cond if $2.maxval,$4.minval ∈ ℤ then $2.maxval < $4.minval
              else true
       $$.minval := if $2.minval ∈ ℤ then $2.minval
              else if $2.maxval ∈ ℤ then $2.maxval
              else if $4.minval ∈ ℤ then $4.minval else $4.maxval
       $$.maxval := if $4.maxval ∈ ℤ then $4.maxval
              else if $4.minval ∈ ℤ then $4.minval
              else if $2.maxval ∈ ℤ then $2.maxval else $2.minval
command    = "if" expression "then" cmd_seq "end".
       $$.assigncount := $4.assigncount
       $$.ifcount := $4.ifcount + 1
       cond if $2.maxval,$4.minval ∈ ℤ then $2.maxval < $4.minval
              else true
       $$.minval := if $2.minval ∈ ℤ then $2.minval
              else if $2.maxval ∈ ℤ then $2.maxval
              else if $4.minval ∈ ℤ then $4.minval else $4.maxval
```

```
      $$.maxval := if $4.maxval ∈ ℤ then $4.maxval
            else if $4.minval ∈ ℤ then $4.minval
            else if $2.maxval ∈ ℤ then $2.maxval else $2.minval
command    = "if" expression "then" cmd_seq "else" cmd_seq "end".
      $$.assigncount := $4.assigncount + $6.assigncount
      $$.ifcount := $4.ifcount + $6.ifcount + 1
      cond if $2.maxval,$4.minval ∈ ℤ then $2.maxval < $4.minval
            else true
      cond if $4.maxval,$6.minval ∈ ℤ then $4.maxval < $6,minval
            else true
      $$.minval := if $2.minval ∈ ℤ then $2.minval
            else if $2.maxval ∈ ℤ then $2.maxval
            else if $4.minval ∈ ℤ then $4.minval
            else if $4.maxval ∈ ℤ then $4.maxval
            else if $6.minval ∈ ℤ then $6.minval else $6.maxval
      $$.maxval := if $6.maxval ∈ ℤ then $6.maxval
            else if $6.minval ∈ ℤ then $6.minval
            else if $4.maxval ∈ ℤ then $4.maxval
            else if $4.minval ∈ ℤ then $4.minval
            else if $2.maxval ∈ ℤ then $2.maxval else $2.minval
cmd_seq    = command.
      $$.assigncount := $1.assigncount
      $$.ifcount := $1.ifcount
      $$.minval := $1.minval
      $$.maxval := $1.maxval
cmd_seq    = cmd_seq ";" command.
      $$.assigncount := $1.assigncount + $3.assigncount
      $$.ifcount := $1.ifcount + $3.ifcount
      cond if $2.maxval,$4.minval ∈ ℤ then $2.maxval < $4.minval
            else true
      $$.minval := if $2.minval ∈ ℤ then $2.minval
            else if $2.maxval ∈ ℤ then $2.maxval
            else if $4.minval ∈ ℤ then $4.minval else $4.maxval
      $$.maxval := if $4.maxval ∈ ℤ then $4.maxval
            else if $4.minval ∈ ℤ then $4.minval
            else if $2.maxval ∈ ℤ then $2.maxval else $2.minval
expression = term.
      $$.minval := $1.minval
      $$.maxval := $1.maxval
expression = expression ("+" | "-" | "OR") term.
      cond if $1.maxval,$3.minval ∈ ℤ then $1.maxval < $3.minval
            else true
      $$.minval := if $1.minval ∈ ℤ then $1.minval
            else if $1.maxval ∈ ℤ then $1.maxval
            else if $2.minval ∈ ℤ then $2.minval else $2.maxval
      $$.maxval := if $2.maxval ∈ ℤ then $2.maxval
            else if $2.minval ∈ ℤ then $2.minval
            else if $1.maxval ∈ ℤ then $1.maxval else $1.minval
term       = factor.
      $$.minval := $1.minval
      $$.maxval := $1.maxval
term       = term ("*" | "/" | "AND") factor.
      cond if $1.maxval,$3.minval ∈ ℤ then $1.maxval < $3.minval
            else true
      $$.minval := if $1.minval ∈ ℤ then $1.minval
            else if $1.maxval ∈ ℤ then $1.maxval
            else if $2.minval ∈ ℤ then $2.minval else $2.maxval
      $$.maxval := if $2.maxval ∈ ℤ then $2.maxval
            else if $2.minval ∈ ℤ then $2.minval
```

```
              else if $1.maxval ∈ ℤ then $1.maxval else $1.minval
factor     = "NOT" factor | "(" expression ")".
       $$.minval := $2.minval
       $$.maxval := $2.maxval
factor     = "number".
       $$.minval := $1.lexval
       $$.maxval := $1.lexval
factor     = "identifier" | "true" | "false".
       $$.minval := NaN
       $$.maxval := NaN
factor     = expression ("<"|">"|"<="|">="|"<>"|"=") expression
       cond if $1.maxval,$3.minval ∈ ℤ then $1.maxval < $3.minval
              else true
       $$.minval := if $1.minval ∈ ℤ then $1.minval
              else if $1.maxval ∈ ℤ then $1.maxval
              else if $2.minval ∈ ℤ then $2.minval else $2.maxval
       $$.maxval := if $2.maxval ∈ ℤ then $2.maxval
              else if $2.minval ∈ ℤ then $2.minval
              else if $1.maxval ∈ ℤ then $1.maxval else $1.minval
```

3.

Ein Problem dürfte in jedem Falle sein, dass *jedes* Attribut, welches *irgendwo* im Programm verwendet wird, in der „Wurzel“ des jeweiligen Regelteilbaumes vorhanden sein muss. Das erzeugt schon einmal einen ziemlichen Wust von Attributen, die der Übersichtlichkeit wohl nicht förderlich sind.

Weiterhin dürften Sprachen, die sinnvolle Einschränkungen über semantische Regeln in der Grammatik beschreiben wollen, nicht sehr komplex werden können, wenn nur erbende Regeln möglich sind, da es oft nötig ist, einen Datenfluss von unten nach oben zu haben. Beispielsweise in Sprachen wie W oder Pascal benötigt man eine Deklaration der Variablen vor ihrer Benutzung. Während man die deklarierten Variablen noch zum jeweiligen Codeblock weiterreichen kann über eine erbende Regel, so kann man aus dem Inhalt des Blockes selbst keine deklarierten Variablen an den Block weiterreichen. Folglich weiß „declaration“ nichts über die deklarierten Variablen. Eigentlich ein KO-Argument gegen den Nutzen, weshalb wir Attributgrammatiken überhaupt eingeführt hatten.

4.

Grundidee hier ist, dass jede Nicht-Primzahl sich als Produkt zweier natürlicher Zahlen $n, m \geq 2$. Zunächst hier also die Metagrammatik, die mit dem Symbol N eine natürliche Zahl in unärer Darstellung liefert:

$$G_M = (\{A, N\}, \{a\}, \{(N, a), (N, Na), (A, NN)\}, A)$$

Nun benötigen wir noch die Modellregeln mit dem Startsymbol S, die unsere eigentliche Grammatik erzeugen:

$$S \to XX$$
$$X \to N$$
$$X \to NX$$