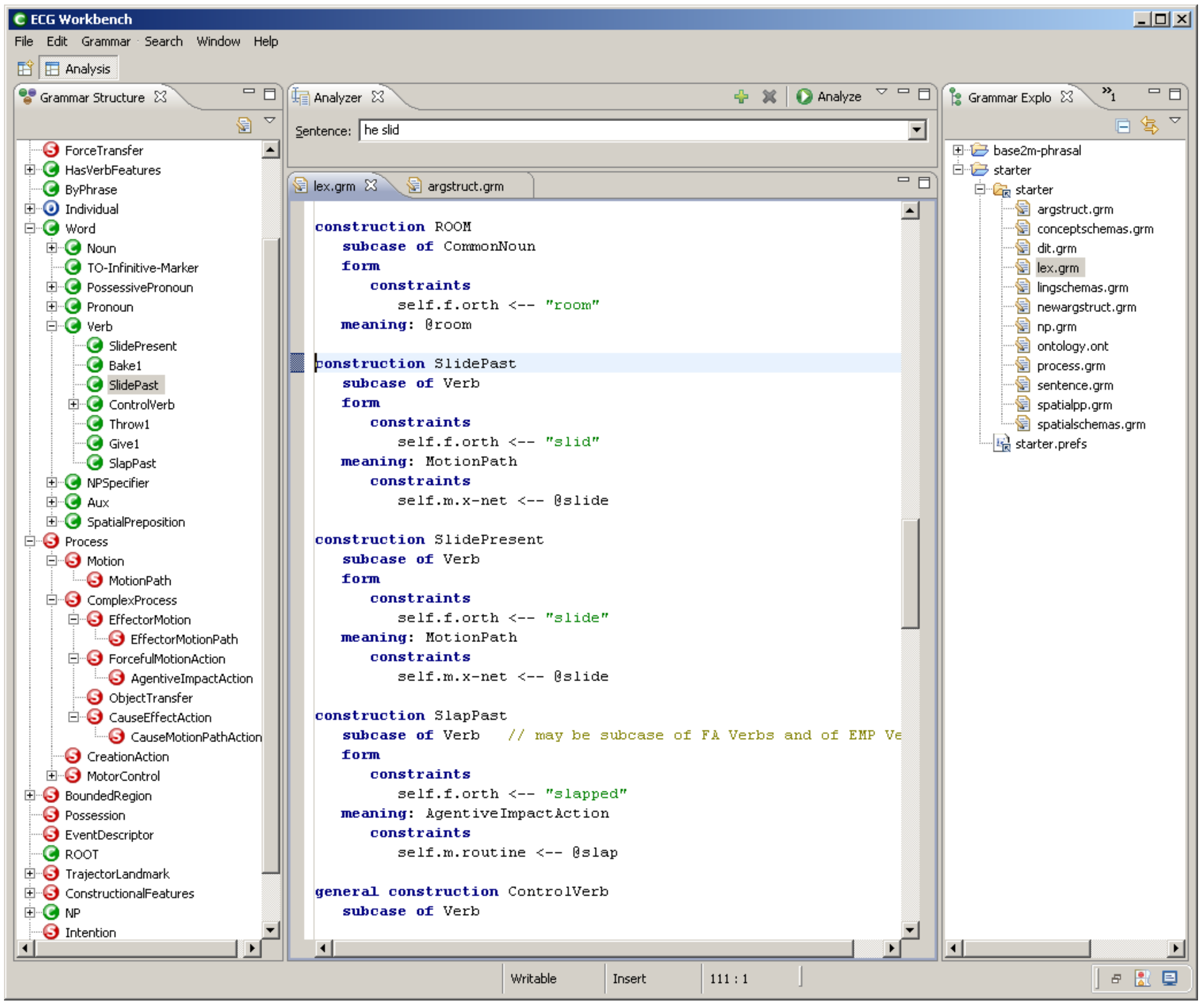
ECG Workbench (draft, v. 0.3)

The ECG Workbench (WB for short) fulfills two related functions. One is the creation and maintenance of grammars written in the ECG formalism. The other is as a testing tool for such grammars. The main aim is to simplify the operations of creating, testing, and revising ECG grammars for the linguist.

Unification grammars in general and ECG grammars in particular are sets of very tightly-coupled rules: to successfully master them requires the ability to recognize how a change in one part can affect other parts and the grammar as a whole. As described in <BD>, the basic components of ECG, constructions and schemas, are organized as subcase lattices—hierarchical inheritance structures with multiple parents. The long column on the left of Figure 1 depicts a portion of the lattices for an example grammar that we will discuss in this chapter. One can see that the **SlidePast** construction is a subcase of **Verb**, which is a subcase of **Word**, which is a subcase of **RootType**.

In order to provide the flavor of the kind of aid the Workbench affords the grammarian, we will show a few examples of how the tool can be used for analyzing a sentence licensed by a simple grammar. We will analyze a simple sentence in the Workbench. Figure 1 is how the latest Workbench (version 0.6 at the time of this writing) typically looks when used to examine a grammar file.



```
construction ROOM
   subcase of CommonNoun
   form
      constraints
         self.f.orth <-- "room"
   meaning: @room

construction SlidePast
   subcase of Verb
   form
      constraints
         self.f.orth <-- "slid"
   meaning: MotionPath
      constraints
         self.m.x-net <-- @slide

construction SlidePresent
   subcase of Verb
   form
      constraints
         self.f.orth <-- "slide"
   meaning: MotionPath
      constraints
         self.m.x-net <-- @slide

construction SlapPast
   subcase of Verb   // may be subcase of FA Verbs and of EMP Ve
   form
      constraints
         self.f.orth <-- "slapped"
   meaning: AgentiveImpactAction
      constraints
         self.m.routine <-- @slap

general construction ControlVerb
   subcase of Verb
```

Figure 1

The Workbench application has only one window, which is subdivided in various tabbed areas, called views and editors. Figure 1 shows three main areas. As already mentioned above, to the left **Grammar Structure** summarizes the lattices that structure constructions, schemas, and also the ontological model (explained below). The view to the right **Grammar Explo(rer)**, displays all the files comprising the grammar in use.

The central part contains one of the actual grammar description files: **verbs.grm,** in which the construction for **SlidePast** is highlighted. Clicking on a node in the left-hand pane automatically brings up the grammar unit containing the clicked node, highlighting the definition for it. The breakdown of a grammar into files is only for the grammarian's convenience, as neither ECG nor the WB imposes any constraints on that. The WB allows one to add new schemas and constructions directly by adding definitions in the central pane and these are automatically added to the lattice representation on the left.

After a grammar is modified, it can be checked for form and meaning consistency. When the grammar consistency checking routine finds errors in constructions or schemas, it marks them by underlining the constructs that caused a complaint from the consistency checker. The details of the complaint can be seen by hovering the mouse point over the underlined element. To help with multiple file grammars, error complaints are also marked on each unit in the column at the right-hand side. Details of all the errors can be seen by clicking on the unit nodes, or by looking at the **Problems** view (not shown) which lists them all along with a detailed description.

Sentence Analysis

As explained above, one of the uses of the software is to create and maintain grammars. More interestingly, the Workbench's primary use is for analyzing sentences. This is achieved by entering a sentence (e.g., "he slid") in the narrow **Analyzer** narrow window in the upper center of (Figure 1).

The detailed analysis is carried out by a separate Analyzer program that will be discussed in the remainder of this chapter. The sentence analyzer's output is currently available in two alternative forms. The first is completely textual. For each analysis, this shows the cost, the constructions and schemas used in the analysis, the semantic constraints, and a semantic specification (or SemSpec). More precisely, the textual output consists of the following:

- Cost: As we will describe in this chapter, the underlying analyzer uses sophisticated numerical scoring to find the best syntactic and semantic fit for the given input.
- Constructions used: the current implementation of the ECG Analyzer employs a partially generative model for the syntactic part of the analysis. This section lists the constructions along with their span in parentheses (Figure 2). The numbers in square brackets are arbitrary, but are used to denote matching elements (bindings). For example **SlidePast** covers positions 1 to 2 in the input and has code [22]. The indentation is not part of the output, but is shown to emphasize the tree structure rooted at the **ROOT** construction.

```
0 he 1 slid 2
ROOT[2] (0, 2)
  Declarative[1] (0, 2)
     He[5] (0, 1)
        IntransitiveArgumentStructure[10] (1, 2)
           SlidePast[22] (1, 2)
```

Figure 2: Tree with Constructions used.

- Schemas used: the list of all the schemas used in the semantic part of the analysis, and also elements in the ontology, prefixed by the **@** character.

```
EventDescriptor[1]
Finite[4]
VerbFeatureSet[7]
NominalFeatureSet[8]
@entity[10]
MotionPath[3]
RD[6]
@maleAnimate[13]
@slide[21]
SPG[24]
```

Figure 3: Schemas used

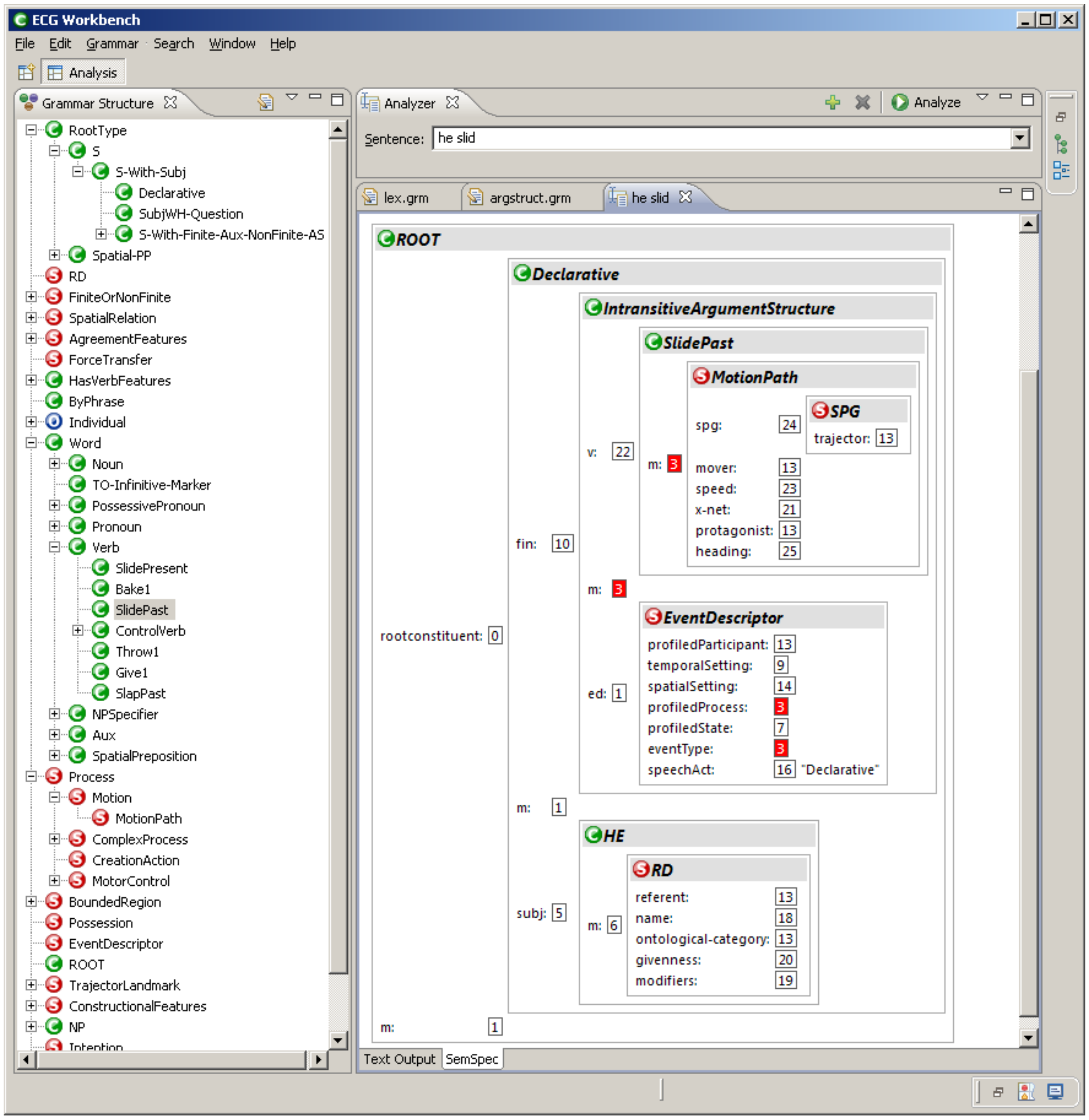
- Semantic constraints: the list of all the bindings that took place in the analysis. Each block shows bound roles; the last element is the common filler. For example, for the case in Figure 4, the schema **MotionPath** is the common filler for the roles **eventType** and **profiledProcess** of an **EventDescriptor** schema instance, the meaning poles of two constructions, **IntransitiveArgumentStructure** and **SlidePast**. The fact that these six elements are bound together, as indicated by the double

```
EventDescriptor[1].eventType ↔
EventDescriptor[1].profiledProcess ↔
IntransitiveArgumentStructure[10].m ↔
SlidePast[22].m
  Filler: MotionPath[3]
```

Figure 4: Partial bindings.

arrows (↔**),** represents coindexation: all of them are assigned the same index as their common filler (which is 3 in this case) by the analysis process, as will be explained.

- The semantic specification: this is a textual representation of the resulting analysis structure (SemSpec). Numbers in square brackets index instances, the same ones listed in the two lists above (constructions and schemas), and also shown as boxed number in the graphical version, shown below, Figure 5. The headers are marked in the same way as in the tree-like view on the left side of fig. 1 above: discs with a "C" label mark constructions, ones with an "S" mark schemas. Clicking on the boxed indices lights up all those that labels with a common binding. Figure 5 shows the same situation described in Figure 4: index [3] denotes the roles and meaning poles listed above in Semantic constraints.



ECG Workbench

File Edit Grammar Search Window Help

Analysis

Grammar Structure

```
RootType
  S
    S-With-Subj
      Declarative
      SubjWH-Question
      S-With-Finite-Aux-NonFinite-AS
  Spatial-PP
RD
FiniteOrNonFinite
SpatialRelation
AgreementFeatures
ForceTransfer
HasVerbFeatures
ByPhrase
Individual
Word
  Noun
  TO-Infinitive-Marker
  PossessivePronoun
  Pronoun
  Verb
    SlidePresent
    Bake1
    SlidePast
    ControlVerb
    Throw1
    Give1
    SlapPast
  NPSpecifier
  Aux
  SpatialPreposition
Process
  Motion
    MotionPath
  ComplexProcess
  CreationAction
  MotorControl
BoundedRegion
Possession
EventDescriptor
ROOT
TrajectorLandmark
ConstructionalFeatures
NP
Intention
```

Analyzer — Analyze

Sentence: he slid

lex.grm | argstruct.grm | he slid

```
ROOT
rootconstituent: [0]
  Declarative
  fin: [10]
    v: [22]
      IntransitiveArgumentStructure
      m: [3]
        SlidePast
          MotionPath
            spg:        [24]   SPG
                               trajector: [13]
            mover:      [13]
            speed:      [23]
            x-net:      [21]
            protagonist:[13]
            heading:    [25]
    m: [3]
    ed: [1]
      EventDescriptor
        profiledParticipant: [13]
        temporalSetting:     [9]
        spatialSetting:      [14]
        profiledProcess:     [3]
        profiledState:       [7]
        eventType:           [3]
        speechAct:           [16] "Declarative"
  m: [1]
  subj: [5]
    HE
    m: [6]
      RD
        referent:            [13]
        name:                [18]
        ontological-category:[13]
        givenness:           [20]
        modifiers:           [19]
m: [1]
```

Text Output | SemSpec

Figure 5

The Grammar

The most important piece of information produced by the Analyzer is the Semantic Specification, or SemSpec. This section describes the main principles and those aspects of the grammar that are involved in the latter.

As in all construction grammars, constructions bind together form constraints and meaning constraints. We will first describe the components of the SemSpec in Fig. 5 and then explain how the WB depicts the complete analysis of the sample sentence.

Schemas, used to represent the meaning constraints of a construction, are embodied semantic schemas. As already pointed out, constructions and schemas, and in general all ECG primitives, are organized in inheritance structures. An inheritance relation (a subtype) is specified in the grammar by the **subcase of** keyword. Other relations are specified by the **roles** keyword, which introduces a part (or feature) in the structure within which it is used, and by the **evokes** keyword, which identifies an evoked structure that is neither a subpart nor a subtype. Again, binding is specified by the double arrows (↔). Finally, comments are signaled by double slashes (//). The figures on the right contain the grammar specification for the semantic schemas involved in the above analysis.

```
schema TrajectorLandmark
  roles
    trajector
    landmark
    profiledArea
```

```
schema SPG
  subcase of TrajectorLandmark
  roles
    source
    path
    goal
```

```
schema Process
  roles
    protagonist
    x-net: @process
```

```
schema Motion
  subcase of Process
  roles
    mover: @entity
    speed    // scale
    heading  // place
    x-net: @motion   // modified
    protagonist      // inherited
constraints
    mover ↔ protagonist
```

```
schema MotionPath
  subcase of Motion
  evokes SPG as spg
  constraints
    mover ↔ spg.trajector
```

TrajectorLandmark and **SPG** (Source-Path-Goal) represent conventional image schemas related by inheritance. That is, SPG inherits all the structure from its supertype: in this case, the roles **trajector**, **landmark**, and **profiledArea**.

The schema for **Process**, and thus the one for **Motion**, describe actions that profile a protagonist. The x-net role is typed (via the colon) to be a kind of x-schematic structure representing a generic **process** (stored in the ontology). X-

Schemas are fine-grained process structure representations. For instance, action like walking or pushing can be represented as x-schematic structure. In the **Motion** schema, the **mover** role is also typed to be a generic **entity**. In the **constraints** section, the **mover** is bound to the **protagonist** role, inherited from **Process**. The **evokes** relation is shown in **MotionPath**, which represents a bounded motion along a path. Such motion is specified to evoke a source-path-goal image schematic structure, made locally available as the **spg** symbol. In the last line the **mover**, inherited from the schema's more abstract supertype **Motion**, is identified with the **trajector** of the evoked source-path-goal image schema. The last two schemas introduce descriptors. One is **EventDescriptor**, which, as described in <DB> in this book, typically represents meaning of an entire scene, as provided by the verbal argument structure (the **eventType** role) and by the verb's meaning itself (the **profiledProcess** role). The second one is for referents (**RD** for Referent Descriptor) and typically represents the constraints associated with the referents of nominal and pronominal constructions.

```
schema EventDescriptor
  roles
    eventType: Process
    profiledProcess: Process
    profiledParticipant
    profiledState
    spatialSetting
    temporalSetting
    speechAct
```

```
schema RD
  roles
    ontological-category
    givenness
    referent
    number
```

With these semantic structures in hand, we can examine the constructions that lead the compositional process generating the analysis shown above in Figure 5. The **Verb** construction takes advantage of multiple inheritance. Its ancestors, **Word** and **HasVerbFeatures**, not shown, define form constraints for words (the fact

```
general construction Verb
  subcase of Word, HasVerbFeatures
  meaning: Process
```

```
construction SLIDEPAST
  subcase of Verb
  form
    constraints
      self.f.orth ← "slid"
  meaning: MotionPath
    constraints
      self.m.x-net ← @slide
```

that a word has a certain graphical or phonetic representation), and for verbal agreement features such as number and person. The **SlidePast** constrains its form and meaning poles, which are referred to via the usual dotted notation by the **f** and **m** pseudoroles respectively: the **orth** role (for orthography) is set to the atomic value "slid" using the left-arrow (←). On the meaning side, the construction's meaning is typed as **MotionPath**, illustrated above. The x-schematic motor program is also set in the **constraints** line to be the **@slide** x-net.

As can be seen from the Construction list shown on the left of Fig. 5, one construction under **ROOT** is the **Declarative**, which spans the whole sentence. It brings together subject, an NP construction that is the supertype of the pronominal constructions like the one for **He** (not shown), and a verb of type **VerbPlusArgument**, of which IntransitiveArgumentStructure, the construction actually chosen by the Analyzer's best fit process.

```
general construction NP
  subcase of RootType
  constructional: NominalFeatures
  meaning: RD
```

```
construction Declarative
  subcase of S-With-Subj
  constructional
    constituents
      subj: NP    // inherited
      fin: VerbPlusArguments
  form
    constraints
      subj.f before fin.f
  meaning
    constraints
      // inherited
          subj.m.referent ↔ self.m.profiledParticipant
      self.m ↔ fin.ed
      self.m.speechAct ← "Declarative"
```

```
general construction ArgumentStructure
  subcase of HasVerbFeatures
  meaning: Process
      evokes EventDescriptor as ed
      constraints
        self.m ↔ ed.eventType
```

```
general construction VerbPlusArguments
  subcase of ArgumentStructure
  constructional
      constituents
        v: Verb
      constraints
        self.features ↔  v.features
  meaning: Process  // inherited
    constraints
      v.m ↔ ed.profiledProcess
      evokes EventDescriptor as ed // inher'd
      self.m ↔ ed.eventType  // inherited
```

The elements of the SemSpec are then bound together as follows. The **VerbPlusArgument** construction binds the Verb's meaning pole with the evoked Event

Descriptor's **profiledProcess**. At the same time it binds its own meaning pole (**self.m**) with the **ED**'s **eventType**. The Declarative construction finally binds that this same Event Descriptor to its meaning pole. Besides, it also constrains the subject's **referent** (a role of the Referent Desciptor schema, see above) to be the same as its **profiledParticipant** role. At the form side, it simply constrains the subject to come **before** the verb.

```
construction IntransitiveArgumentStructure
  subcase of VerbPlusArguments
  constructional
      constituents                        // inherited
        v: Verb                           // inherited
      constraints                         // inherited
        self.features ↔  v.features       // inherited
      constraints
        self.features.verbform ⟵ FiniteOrGerund
  meaning: Process
    constraints
      self.m.protagonist ↔ ed.profiledParticipant
      self.m ↔ v.m
      evokes EventDescriptor as ed // inher'd
      self.m ↔ ed.eventType  // inherited
```

The last piece of the analysis is the argument structure chosen by the best fit process: **IntransitiveArgumentStructure**. This constrains the protagonist of the action, or of the motion in this case, to be the **EventDescriptor**'s **profiledParticipant**. The **EventDescriptor** structure represented by **ed**, inherited from **VerbPlusArguments**. In the last line it also says that its meaning is the verb's meaning. This, together with the constraint **v.m ↔ ed.profiledProcess** described above for **VerbPlusArguments**, implies that the even described by the intransitive argument structure is the same as the one described by the verb (see the description above for the **EventDescriptor** schema).