

# Crossroads

## A Crossword Compiler and Solver

Andrea Reale

DEIS  
Università di Bologna

Linguaggi e Modelli Computazionali LS  
7 Aprile 2009

# Outline

- 1 Introduction
- 2 The Grammar
- 3 Implementation
- 4 Configuration
- 5 Conclusions and future work

# Crossroads

## Main ideas

- Environment for the description and solution of crosswords
- Allows to
  - ▶ Describe crossword instances
  - ▶ Export the given description in several representations
- Offers simple tools for crossword solving

# Goals

and requirements

## Requirements

- Ease of use
  - ▶ Intuitive syntax, as close as possible to natural speech
  - ▶ Minimum language complexity
- Modularity and expandability
  - ▶ Several formats to export to
  - ▶ Ability to use interchangeable components for each part of the system

# What is a Crossword

and how to describe one

## Definition (from Apple Dictionary)

*a puzzle consisting of a grid of squares and blanks into which words crossing vertically and horizontally are written according to clues.*

- Not so punctual definition
- Need for a more precise model to describe a crossword
- We will consider only crosswords with rectangular  
schemarettangolare

# The Model

## Dimensions and schema

### Definition

#### Dimensions

- Couple  $(m, n) \in \mathbb{N} \times \mathbb{N}$
- The number of *righe* and *columns* of the crossword, respectively

#### Schema

- Determined by the function:

$$b : \bar{m} \times \bar{n} \rightarrow \{0, 1\}$$

- Maps each cell of the crossword to 0 if it is white, to 1 if it is black

# The Model

## Words

### Definition

#### Entry

- A set  $S \subset \bar{m} \times \bar{n}$
- It is made of the sequence of cells - horizontally or vertically aligned - that are not interrupted by black cells
- i.e. each entry marks a set of cells capable to contain a word

#### Dictionary

- A set  $D$  made of strings of a language  $L \subseteq \Sigma^*$

# The Model

## Words

### Definition

- Word**
- Given an entry  $S_i$  and a dictionary  $D_i$ , a word is a function:

$$p : S_i \rightarrow \Sigma$$

- Maps each cell to a symbol, in such a way that the concatenation of those symbols is an element  $D_i$

- Solution**
- Maps each possible entry  $S_i$  of a schema to a word  $p_{S_i}$



# The Model

## Solution

### Valid Solution

- Let  $X$  be the set of couples of entries having non-empty intersection
  - ▶ Notice that the intersection will contain at most one element
- A valid solution maps to each couple of entries  $(S_i, S_j) \in X$  to the words  $p_{S_i}$  e  $p_{S_j}$  such that:

$$p_{S_i}(r, c) = p_{S_j}(r, c)$$

where  $S_i \cap S_j = \{(r, c)\}$

# “Traditional” Crossword

## What's different

- “Normal” crossword have just one dictionary for every entry
  - ▶ That is the dictionary of the natural language of the crossword
  - ▶ plus some special string (e.g. *in the middle of the ocean*);
- Presence of **definitions**
  - ▶ Hints about the *entries-words* association
  - ▶ On the theory they are not strictly necessary to find a solution
  - ▶ They are used to shrink the set of possible solutions

# Separation of concerns

- Two degrees of separation
- **Description vs Solution**
  - ▶ A language to describe a crossword
  - ▶ A language to describe how to solve it (Prolog)
- **Schema vs Definitions**
  - ▶ A language to describe the schema
  - ▶ A language to specify the definitions associated to entries

# How it looks like

## Schema

### Example (cross1.schema)

```
crossword exampleCwd:
  size is 5 x 5
  holes schema:
    4
    1 3 4
    none
    0 1 3
    0
end
```

<b>1</b>		<b>2</b>		
<b>3</b>				<b>4</b>
	<b>5</b>			

# How it looks like

## Definitions

### Example (cross1.defs)

```
crossword exampleCwd:
  definitions language is it:
    across:
      def "National Aeronautics and Space Administration"
      def "Skateboarding trick"
      def "Il Nicolas protagonista di Con Air"
    down:
      def "Thomas Anderson in Matrix"
      def "Lo e` il tabasco"
      def "Netbook ASUS"
end
```

**Across:**

1. National Aeronautics and Space Administration
3. Skateboarding trick
5. Il Nicolas protagonista di Con Air

**Down:**

1. Thomas Anderson in Matrix
2. Lo e` il tabasco
4. Netbook ASUS

# The Grammar for Schemas

- Header
  - ▶ Crossword id specification
- Dimensions
- Specification of the schema function
  - ▶ A line for each row in the crossword
  - ▶ An natural number for each **hole** in the row→ it points the respective column
  - ▶ *none* if the row is completely white
  - ▶ The last lines can be omitted if the corresponding rows are completely white
- Necessary (and sufficient) information to specify the schema

# Excerpts of the grammar

## Schemas grammar

### Example (Excerpts)

```

<SCHEMA_FILE> ::= <HEAD><INDENT><SCHEMA_DEF><DEDENT><END>
<HEAD> ::= "crossword" <ID> ":" <NL>
<SCHEMA_DEF> ::= <DIM_DEF><HOLES_SCHEMA>
<DIM_DEF> ::= "size" "is" <NATURAL> "x" <NATURAL> <NL>
<HOLES_SCHEMA> ::= "holes" "schema" ":" <NL> <INDENT>
                  <HOLES_LINES><DEDENT>
<HOLES_LINES> ::= <LINE><NL> | <LINE><NL><HOLES_LINES>
<LINE> ::= "none" | <HOLES_SEQ>
...

```

# The Grammar for Definitions

- Header
  - ▶ Crossword id specification
- List of definitions
  - ▶ Ability to specify localized definitions
  - ▶ A set o definitions for each language
  - ▶ Is it useful?
- Separation of across and down definitions



# Excerpts from the grammar

## Definitions grammar

### Example (Excerpts)

```

<DEFS_FILE>          ::= <HEAD><INDENT><DEFS_SKEL><DEDENT><END><NL>
<HEAD>               ::= "crossword" <ID> ":" <NL>
<DEFS_SKEL>          ::= <UNLOCALIZED_BODY> | <LOCALIZED_BODIES>
...
<UNLOCALIZED_BODY>   ::= "definitions" ":" <NL>
                        <INDENT><DEF_BODY><DEDENT>
<LOCALIZED_BODY>     ::= "definitions" "language" "is"
                        <LANG_CODE> ":" <NL>
                        <INDENT><DEF_BODY><DEDENT>
<DEF_BODY>           ::= "across" ":" <NL><INDENT><DEFS><DEDENT>
                        "down" ":" <NL><INDENT><DEFS><DEDENT>
...

```

# Considerations about the grammar

## Classification

- **Type 2** Grammar (according to Chomsky's classification)
- Generates a **Type 3** (regular) language
  - ▶ There's not *self-embedding*
  - ▶ Each part (and the whole) grammar could be expressed with a *regex*
- **LL(1)** Grammar
  - ▶ Parsing *top-down*
- No  $\epsilon$ -rules

# Overview

## Putting pieces together

- **Interfaccia CrosswordParser**
  - ▶ Schema/Definitions Parsing
  - ▶ Parsing of sources must produces an instance of the **Crossword** class
- **Crossword Class (Java)**
  - ▶ Describes a crossword instance
  - ▶ It is used to handle and explore the crossword
- **CrosswordExporter Interface**
  - ▶ `public T toExternalRepresentation(Crossword input)`
- **SolvingEngine Interface**
  - ▶ `public void solve(Crossword toSolve)`

# Scanner and Parser

## Implementation

- Prolog Parser
  - ▶ tuProlog with DCGLibrary
  - ▶ Copy/Past of the grammar in a Prolog Theory :)
  - ▶ Builds the AST as a Prolog term
- “Home-made” Scanner written in Java
  - ▶ Generates the atoms for the parser
  - ▶ Handles indentation, and generates <indent>/<dedent> tokens using an internal stack
- The AST is explored with an ad-hoc Prolog predicate
  - ▶ Crossword's parameters are passed to the Java environment
  - ▶ *Unification* makes everything easy

# Correctness tests

- Basic syntax checks made by the Scanner
  - ▶ Presence of not allowed symbols
  - ▶ Correctness of indentation
- Syntactical correctness of single tokens made on the Prolog side
- Semantical errors are found by the constructor of the class Crossword

# Default Exporters

Prolog, SVG, XHTML

- **Prolog Exporter**

- ▶ Exports a crossword in the Prolog theory that solves it
- ▶ It uses Google to build dictionaries for each entry

- **SVG Exporter**

- ▶ Exports a crossword in the vectorial image representing it
- ▶ If the instance is solved, includes the solution
- ▶ With or without definitions

- **XHTML Exporter**

- ▶ Exports a crossword into an XHTML page representing it
- ▶ It uses the SVG Exporter

# Crossword Solver

sometimes it works

- Small dictionaries for each entry
  - ▶ Searches the Web for the definitions using **Google** AJAX Search API
  - ▶ Extracts from the results the most frequent terms of the right length
- Generates the solving theory with a PrologExporter
  - ▶ Uses **tuProlog** to find a solution

# Configuration

## Configuration file

- Set of key-value couples
  - ▶ Specification of classes implementing the core interfaces
  - ▶ Several parameters for the solver
- **JSON** (Javascript Object Notation)
  - ▶ Really simple and efficient notation
  - ▶ Ready to use parsers
- XML-based configuration of **WebHarvest**
  - ▶ Explains how to process web pages
  - ▶ Ability to use regular expressions
  - ▶ Node selection with XPath



# Conclusions

## Limits and possible solutions

- Can't solve *tricky* definitions (e.g. in the middle of something)
  - ▶ Implementation of ad-hoc rules
- Poor parser performance
  - ▶ Code optimization ( by now the parser is written in pure declarative prolog, without cuts)
  - ▶ or Reimplementation of the parser with something better performing (e.g. JavaCC)
- Relevant time used to build dictionaries
  - ▶ Network access times. We have less to do here.

# Conclusions

## Limits and possible solutions

- Solution time is EXPTIME in the number of entries
  - ▶ Smaller dictionaries
  - ▶ Better dictionaries (use of Thesauri, ad-hoc rules)
  - ▶ Smarter resolution theory
- Better error handling
  - ▶ The Prolog parser is not good into giving exhaustive errors description
- Better user interaction
  - ▶ Step-by-step resolution
  - ▶ Interactive GUI
  - ▶ Possibility to accept users' hints

