A SAT-based parser and completer for pictures specified by tiling

Matteo Pradella
Stefano Crespi Reghizzi

Politecnico di Milano & CNR IEIIT-MI

PRIN “Linguaggi Formali e Automi: aspetti matematici e applicativi”, Varese 2006.07.17

Parsing 2D languages

Tiling Systems (TS) [GR97]:
parsing is NP-complete [Lindgren+Moore+Nordahl98].

Hence, parsing Tile Rewriting Grammars (TRG) is NP-complete.

Matz's Context-Free Picture Grammars (CFPG):
parsing a $n \times n$ picture takes $O(n^5)$ time [We06].

Is parsing TS practical?

It does not seem so, but:

in practice, some NP-complete problems are successfully tackled with heuristics.

For example: Model-checking in verification (e.g. SPIN, SMV),
the *Boolean Satisfiability Problem* (SAT) (e.g. zChaff, Sato, MiniSAT).

Main idea: encode the TS parsing problem into SAT.

The encoding

Consider a Tiling System $(\Sigma, \Gamma, \Theta, \pi)$.

The encoding is based on the inverse projection of the input picture $\pi^{-1}(p)$.

Consider a possible inverse projection $q \in \pi^{-1}(p)$:
proposition $Q^a_{(i,j)}$ stands for $q(i,j) = a$.

The first formula represents all possible inverse projections of p:

$$F_1 := \bigwedge_{(i,j) \in [(1,1)..|p|]} \left(\bigvee_{a \in \pi^{-1}(p(i,j))} Q^a_{(i,j)} \right)$$

The second formula states that q must satisfy Θ:

$$F_2 := \bigwedge_{(i,j)\in[(1,1)..|p|]} \left(\bigvee_{\theta\in\Theta} \left(\bigwedge_{h,k\in[0,1]} Q^{\theta(h+1,k+1)}_{(i+h,j+k)} \right) \right)$$

The last formula contains the exclusivity constraints:

$$F_3 := \bigwedge_{(i,j)\in[(1,1)..|p|]} \mathsf{OnlyOne}_{a\in\Gamma} \left(Q^a_{(i,j)} \right)$$

The TS-recognition problem is then encoded as the propositional formula $F_1 \wedge F_2 \wedge F_3$

The tool (SatTS)

The SatTS prototype is a command-line tool: we define the TS and a picture in a file, then we call SatTS and obtain either a suitable inverse projection, or *UNSAT*.

Thanks to F_3 the tool accepts *partially specified* pictures (i.e. with *don't cares* symbols "?"), or also *totally unspecified* pictures (i.e. we know just the size).

SatTS is written in Scheme, so it is possible to use Scheme expressions to define complex tilings in a compact and readable way.

Common set operations (union, intersection, complement) and tiling operations ($B_{2,2}$) are already available. (More on this later).

Example: Chinese boxes on a background

$$\Theta = B_{2,2} \begin{pmatrix} \# & \# & \# & \# & \# & \# & \# & \# \\ \# & & & & & & & \# \\ \# & & \nearrow & \rightarrow & \rightarrow & \searrow & & \# \\ \# & & \uparrow & & & \downarrow & & \# \\ \# & & \uparrow & & & \downarrow & & \# \\ \# & & \nwarrow & \leftarrow & \leftarrow & \swarrow & & \# \\ \# & & & & & & & \# \\ \# & \# & \# & \# & \# & \# & \# & \# \end{pmatrix}$$

$\pi(x) = \blacksquare$, for $x \in \{\nearrow, \searrow, \nwarrow, \swarrow, \uparrow, \downarrow, \leftarrow, \rightarrow\}$;
$\pi(\text{blank}) = \text{blank}$

Example picture:

```
# # # # # # # # # # # # # # # # # # # #
#                                     #
#                     ■ ■ ■ ■ ■ ■ ■   #
#   ■ ■ ■ ■ ■ ? ?     ■           ■   #
#   ■         ? ? ?   ■   ■ ■ ■   ■   #
#   ■         ? ?     ■   ■   ■   ■   #
#   ■   ■ ■ ■   ■     ■   ■   ■   ■   #
#   ■   ■   ■   ■     ■   ■   ■   ■   #
#   ■   ■   ■   ■     ■   ■   ■   ■   #
#   ■   ■ ■ ■   ■     ■   ■ ■ ■   ■   #
#   ■           ■     ■           ■   #
#   ■ ■ ■ ■ ■ ■ ■     ■ ■ ■ ■ ■ ■ ■   #
#                                     #
# # # # # # # # # # # # # # # # # # # #
```

We obtain the output:

```
# # # # # # # # # # # # # # # # # # # #
#                                     #
#                     ↗ → → → → → ↘   #
#   ↗ → → → → → ↘     ↑           ↓   #
#   ↑           ↓     ↑   ↗ → ↘   ↓   #
#   ↑           ↓     ↑   ↑   ↓   ↓   #
#   ↑   ↗ → ↘   ↓     ↑   ↑   ↓   ↓   #
#   ↑   ↑   ↓   ↓     ↑   ↑   ↓   ↓   #
#   ↑   ↑   ↓   ↓     ↑   ↑   ↓   ↓   #
#   ↑   ↖ ← ↙   ↓     ↑   ↖ ← ↙   ↓   #
#   ↑           ↓     ↑           ↓   #
#   ↖ ← ← ← ← ← ↙     ↖ ← ← ← ← ← ↙   #
#                                     #
# # # # # # # # # # # # # # # # # # # #
```

Example: Three colors map coloring

The tool gives freedom to specify the tiling with complex expressions (not just $B_{2,2}$).

Colors $= \{\spadesuit, \bigstar, \clubsuit\}$; Boundary: $\diamondsuit$

$\pi(\diamondsuit) = \diamondsuit$; $\pi(c) = \blacksquare$ for $c \in$ Colors.

$\Theta = \overline{\Theta_1 \cup \Theta_2 \cup \Theta_3}$, where:

$$\Theta_1 = \left\{ \boxed{\begin{matrix} \diamondsuit & \spadesuit \\ \spadesuit & \diamondsuit \end{matrix}}, \boxed{\begin{matrix} \diamondsuit & \bigstar \\ \bigstar & \diamondsuit \end{matrix}}, \boxed{\begin{matrix} \diamondsuit & \clubsuit \\ \clubsuit & \diamondsuit \end{matrix}}, \boxed{\begin{matrix} \spadesuit & \diamondsuit \\ \diamondsuit & \spadesuit \end{matrix}}, \boxed{\begin{matrix} \bigstar & \diamondsuit \\ \diamondsuit & \bigstar \end{matrix}}, \boxed{\begin{matrix} \clubsuit & \diamondsuit \\ \diamondsuit & \clubsuit \end{matrix}} \right\}$$

i.e. two neighbors have the same color

$$\Theta_2 = \left\{ \boxed{\begin{matrix} i & j \\ k & l \end{matrix}} : \begin{matrix} i = \diamondsuit = j \\ \vee \\ k = \diamondsuit = l \\ \vee \\ i = \diamondsuit = k \\ \vee \\ j = \diamondsuit = l \end{matrix} \right\}; \quad \Theta_3 = \left\{ \boxed{\begin{matrix} i & j \\ k & l \end{matrix}} : \begin{matrix} i, j \in \text{Colors} \wedge i \neq j \\ \vee \\ k, l \in \text{Colors} \wedge k \neq l \\ \vee \\ i, k \in \text{Colors} \wedge i \neq k \\ \vee \\ j, l \in \text{Colors} \wedge j \neq l \end{matrix} \right\}$$

(we do not consider straight vertical or horizontal borders for simplicity)

A map:

```
# # # # # # # # # #
# ■ ■ ■ ■ ♢ ■ ■ ■ #
# ■ ■ ■ ♢ ■ ■ ■ ■ #
# ■ ■ ♢ ■ ♢ ■ ■ ■ #
# ■ ♢ ■ ■ ■ ♢ ■ ♢ #
# ♢ ■ ♢ ■ ♢ ■ ♢ ■ #
# ■ ■ ■ ♢ ■ ■ ■ ■ #
# ■ ■ ■ ■ ♢ ■ ■ ■ #
# ■ ■ ■ ♢ ■ ■ ■ ■ #
# # # # # # # # # #
```

SatTS tells us how to color it:

#	#	#	#	#	#	#	#	#	#
#	★	★	★	★	♢	♠	♠	♠	#
#	★	★	★	♢	♠	♠	♠	♠	#
#	★	★	♢	♣	♢	♠	♠	♠	#
#	★	♢	♣	♣	♣	♢	♠	♢	#
#	♢	♠	♢	♣	♢	★	♢	★	#
#	♠	♠	♠	♢	★	★	★	★	#
#	♠	♠	♠	♠	♢	★	★	★	#
#	♠	♠	♠	♢	★	★	★	★	#
#	#	#	#	#	#	#	#	#	#

Example: Chinese boxes (no background)

This is the same language we used to present Tile Rewriting Grammars.

An equivalent TS is the following:

$$\Theta = \left\{ \boxed{\begin{matrix} i & j \\ k & l \end{matrix}} : \begin{array}{c} i \in \{\rightarrow, \nearrow\} \Rightarrow j \in \{\rightarrow, \searrow\} \\ \vee \\ j \in \{\downarrow, \searrow\} \Rightarrow l \in \{\downarrow, \swarrow\} \\ \vee \\ l \in \{\leftarrow, \swarrow\} \Rightarrow k \in \{\leftarrow, \nwarrow\} \\ \vee \\ k \in \{\uparrow, \nwarrow\} \Rightarrow i \in \{\uparrow, \nearrow\} \end{array} \right\}$$

$\pi(x) = \circ$, if $x \in \{\rightarrow, \leftarrow, \uparrow, \downarrow\}$;
$\pi(\nearrow) = \ulcorner$; $\pi(\nwarrow) = \llcorner$; $\pi(\searrow) = \urcorner$; $\pi(\swarrow) = \lrcorner$.

A picture:

```
# # # # # #
# ┌ ○ ○ ┐ #
# ○ ┌ ┐ ○ #
# ○ └ ┘ ○ #
# ○ ┌ ┐ ○ #
# ○ └ ┘ ○ #
# └ ○ ○ ┘ #
# ┌ ○ ○ ┐ #
# ○ ┌ ┐ ○ #
# ○ └ ┘ ○ #
# └ ○ ○ ┘ #
# # # # # #
```

Its inverse projection:

#	#	#	#	#	#
#	$\nearrow$	$\rightarrow$	$\rightarrow$	$\searrow$	#
#	$\uparrow$	$\nearrow$	$\searrow$	$\downarrow$	#
#	$\uparrow$	$\nwarrow$	$\swarrow$	$\downarrow$	#
#	$\uparrow$	$\nearrow$	$\searrow$	$\downarrow$	#
#	$\uparrow$	$\nwarrow$	$\swarrow$	$\downarrow$	#
#	$\nwarrow$	$\leftarrow$	$\leftarrow$	$\swarrow$	#
#	$\nearrow$	$\rightarrow$	$\rightarrow$	$\searrow$	#
#	$\uparrow$	$\nearrow$	$\searrow$	$\downarrow$	#
#	$\uparrow$	$\nwarrow$	$\swarrow$	$\downarrow$	#
#	$\nwarrow$	$\leftarrow$	$\leftarrow$	$\swarrow$	#
#	#	#	#	#	#

Example: Chinese blobs

This language is analogous to the Chinese boxes with a blank background. The main difference is that the shape of the box can be anything.

$\pi(x) = \blacksquare$, if $x \in \{\nearrow, \searrow, \nwarrow, \swarrow\}$;
$\pi(\mathsf{blank}) = \mathsf{blank}$.

$\Theta = \Theta_1 \cup \Theta_2$, where:

$$\Theta_1 = \left\{ \boxed{\begin{array}{cc} i & j \\ k & l \end{array}} : \begin{array}{c} i = \searrow \Rightarrow \text{OnlyOne}(j = \nearrow, k = \swarrow, l = \searrow) \\ \vee \\ i = \nwarrow \Rightarrow \text{OnlyOne}(j = \swarrow, k = \nearrow, l = \nwarrow) \\ \vee \\ j = \swarrow \Rightarrow \text{OnlyOne}(i = \nwarrow, k = \swarrow, l = \searrow) \\ \vee \\ \ldots \end{array} \right\}$$

$$\Theta_2 = \left\{ \boxed{\begin{array}{cc} \searrow & \nearrow \\ \swarrow & \nwarrow \end{array}}, \boxed{\begin{array}{cc} \nwarrow & \swarrow \\ \nearrow & \searrow \end{array}}, \boxed{\begin{array}{cc} \searrow & \nearrow \\ \nearrow & \searrow \end{array}}, \boxed{\begin{array}{cc} \nwarrow & \swarrow \\ \swarrow & \nwarrow \end{array}}, \boxed{\begin{array}{cc} \nearrow & \nwarrow \\ \searrow & \swarrow \end{array}}, \boxed{\begin{array}{cc} \swarrow & \searrow \\ \nwarrow & \nearrow \end{array}} \right\}$$

Is this a correct Chinese blob?

```
# # # # # # # # # # # #
#                     #
#                     #
#       ■ ■ ■ ■       #
#     ■       ■       #
#   ■       ■ ■ ■     #
#   ■ ■ ■ ■ ■   ■     #
#     ■ ■ ■ ■ ■ ■     #
#     ■   ■   ■ ■     #
#       ■ ■ ■ ■       #
#         ■ ■         #
#                     #
#                     #
# # # # # # # # # # # #
```

Yes:

```
# # # # # # # # # # # #
#                     #
#                     #
#       ↙ ↖ ↙ ↖       #
#     ↙       ↗       #
#   ↙       ↗ ↗ ↘     #
#   ↘ ↗ ↘ ↗ ↗   ↙     #
#     ↗ ↘ ↗ ↙ ↖ ↘     #
#     ↖   ↙   ↗ ↙     #
#       ↖ ↘ ↗ ↙       #
#         ↖ ↙         #
#                     #
#                     #
# # # # # # # # # # # #
```

Conclusions

The SatTS prototype is fast enough to experiment on reasonably sized (e.g. 200×200) samples

It offers convenient ways for specifying tiles

It has picture completion capabilities

Future: perhaps error correction