



Fast, Flexible (and Inefficient?) Message Parsing

Randolph M. Jones
Soar Technology, Inc.
rjones@soartech.com

Soar Workshop, June 2005

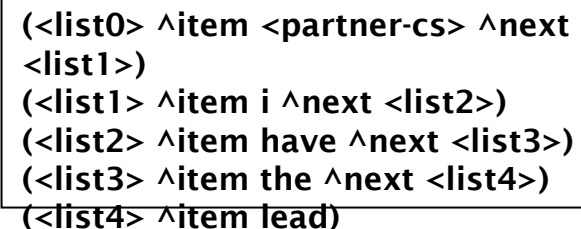
Challenges

- We see increasing demand for interactive intelligent agents that can interact via speech
- We do not have the time or expertise to do full natural language processing
- Neither do our agents
- Agent message processing must be as rich, fast, and flexible as we can make it
 - And it would be nice if the code doesn't become a mess

Background: Message processing in TacAir-Soar

- Template-oriented productions that match against a linear, linked list of message tokens

```
sp "top-ps*persistent*section*tactical*wing-has-the-lead
:o-support
(state <s> ^problem-space.name top-ps
    ^operator <o>
    ^call-sign <my-cs>
    ^command.primary-group <pg>
    ^comm.message <m>)
(<m> ^content <list0> -^processed ^accept *yes*)
(<pg> ^mission.partner.call-sign { <partner-cs> <> <my-cs> }
    ^tactical <st>
    -^type vehicle)
(<st> ^role lead)
[soarList <partner-cs> i have the lead!
-->
(write (crlf) |Wing confirmed taking over as lead|)
(<st> ^role lead - subordinate +)
(<m> ^processed *yes*)
"
```



```
(<list0> ^item <partner-cs> ^next
<list1>)
(<list1> ^item i ^next <list2>)
(<list2> ^item have ^next <list3>)
(<list3> ^item the ^next <list4>)
(<list4> ^item lead)
```

Problems with templates

- Mapping m different templates for a family of messages with n different supplemental conditions can lead to $m \times n$ productions
 - We can address this in part by building intermediate semantic representations of the messages
- For generality messages can contain numerous variables, but that can lead to problems of ambiguity
 - Extreme example from actual TacAir-Soar code:
 - [soarList <x0> <x1> <x2> <x3> <x4> <x5> <x6> <x7> <x8>]
- Templates assume that each message will be received at once and in its entirety

Resolving ambiguity

- In TacAir-Soar, we decided to do a small amount of data-driven tagging in order to resolve such ambiguities

```
[soarList <x0> <x1> <x2> <x3> <x4> <x5> <x6> <x7> <x8>]
```

```
(<list0> ^integer *yes*)
```

```
(<list1> ^integer *yes*)
```

```
(<list2> ^uppercase-alpha *yes*)
```

```
(<list3> ^uppercase-alpha *yes*)
```

```
(<list4> ^uppercase-alpha *yes*)
```

```
(<list5> ^integer *yes*)
```

```
(<list6> ^integer *yes*)
```

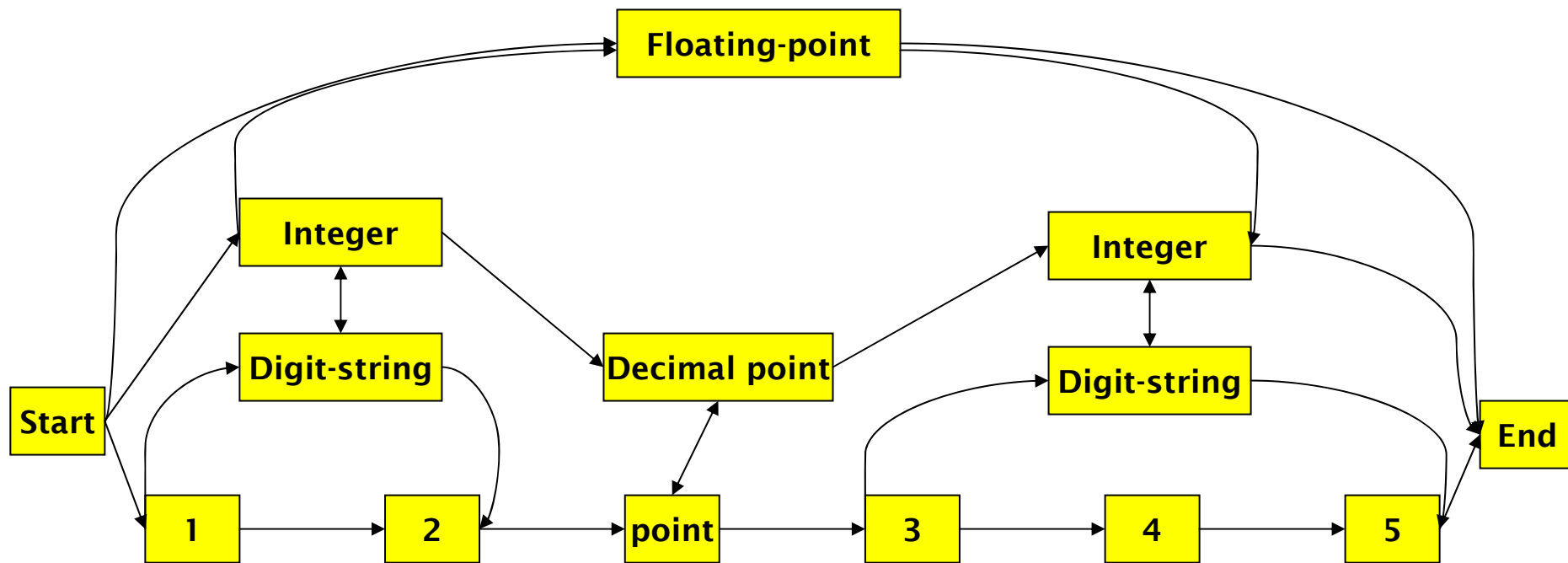
```
(<list7> ^integer *yes*)
```

```
(<list8> ^integer *yes*)
```

```
sp {top-ps*persistent*comm*examine-current-  
item  
  (state <s> ^problem-space.name top-ps  
    ^operator.name  
    ^comm.message <msg>)  
  (<msg> ^message-item <ptr>)  
  (<ptr> ^item <item>)  
-->  
  (<ptr> ^integer (integerp <item>)  
    ^number (numberp <item>)  
    ^literal (literalp <item>))  
}
```

Generalizing the approach

- Data-driven tagging can be used to create a hierarchy of interpretations of the input stream of tokens



Example parsing production

```
sp "top-ps*elaborate*parsing*channel*parse-element*parse*mgrs
  (state <s> ^problem-space.name top-ps
    ^comm.parsing.channel <pc>)
  [soarParseList {^type digit-string ^length << 1 2 >> ^value <zone-number>} \
    {^type single-letter ^value <zone-letter>} \
    {^type single-letter ^value <id-letter1>} \
    {^type single-letter ^value <id-letter2>} \
    {^type digit-string ^string-value <coords>}}]
-->
...
```

Comments on the approach

- Quick (but hopefully not dirty) solution to an urgent problem
 - Need to look to things like NL-Soar for improvements and lessons
- Focus on creating a manageable, reusable module
 - Use elaborations as much as possible
 - Make sure operators are Soar7- and Soar8-friendly
 - Use TCL templates to allow variations in primitive representation of token streams

Advantages

- Allows flexible translation of a variety of syntaxes into hierarchical layers of interpretation
- Generalizes the notion of a “token”, allowing more flexibility in the lengths of messages
 - Also makes it easier to handle individual linked lists that represent only a fragment of a message or contain multiple messages
- Works very naturally within Soar’s elaboration cycle

Potential disadvantages

- Things can get expensive if you let them get out of hand
 - The nature of data-driven processes is that you lose the advantage of top-down control to focus processing
 - The first implementation reached max-elaborations *a lot*
 - And it gets worse the longer the messages are and the more parsing types you need
 - However, Soar makes it possible to fold in context, goals, and attention mechanisms
- There are still potential problems with ambiguity during parsing
 - Can lead to an explosion of interpretation of the input
 - Have to be careful about resolving messages where one potential message is a prefix of another