Generalized Chunking

Mazin Assanie
University of Michigan
mazina@umich.edu

Talk Overview

1. What is chunking?
2. What is generalized chunking?
3. Chunking terminology
4. How Soar 9.3 learns chunks
5. How Soar 9.4 learns chunks

What is chunking?

- Automatic learning mechanism that creates generalized rules which summarize problem-solving in a *substate*.
- These “chunks” will fire in future *similar* situations avoiding similar problem-solving.

When and How?

1. Agent doesn't know how to proceed.
2. Soar creates a substate so agent can consider problem.
3. Agent does problem-solving on the substate and records a result in the superstate.
4. Soar analyzes the problem-solving and creates a generalized rule that summarizes what was needed to produce the result.

What is Generalized Chunking?

What is Generalized Chunking?

- An expansion of Soar's chunking mechanism that creates chunks that are far more *general* and apply to a wider variety of situations.
  - Previous versions of Soar made more specific chunks that contained the exact numeric and string values that occurred when the chunks formed.

What is Generalized Chunking?

- Generalized chunking *analyzes the relationships between* numbers, strings and other *constants* used during problem-solving to **variablize any symbol type**, not just identifiers.

Comparison of Chunks

SOAR 9.3.3

```
sp {chunk*apply*grade
   (state <s1> ^passing-score 75
               ^superstate    nil
               ^student-info <s2>
               ^me-info  { <> <s2> <m1> })
    (<s2>      ^test-score   92
               ^name         Mary)
    -->
    (<s1>      ^decision     <d1>)
    (<d1>      ^name         Mary
               ^score        92
               ^grade        PASS)}
```

SOAR 9.5

```
sp {chunk*apply*grade*9.4
   (state <s1> ^passing-score <p1>
               ^superstate    nil
               ^student-info  <s2>
               ^me-info   { <> <s2> <m1> })
    (<s2>      ^test-score   { > <p1> <s3> }
               ^name          <n1>)
    -->
    (<s1>     ^decision       <d1>)
    (<d1>     ^name           <n1>
              ^score          <s3>
              ^grade          PASS)}
```

Symbol Types: **Constant** **<Variable>**

Implications of this Change

- We expect chunks to be more applicable. They will be *more general but not over-general*.
- We expect agents will need to learn *fewer* chunks.
- We expect agents will learn useful chunks *sooner*.

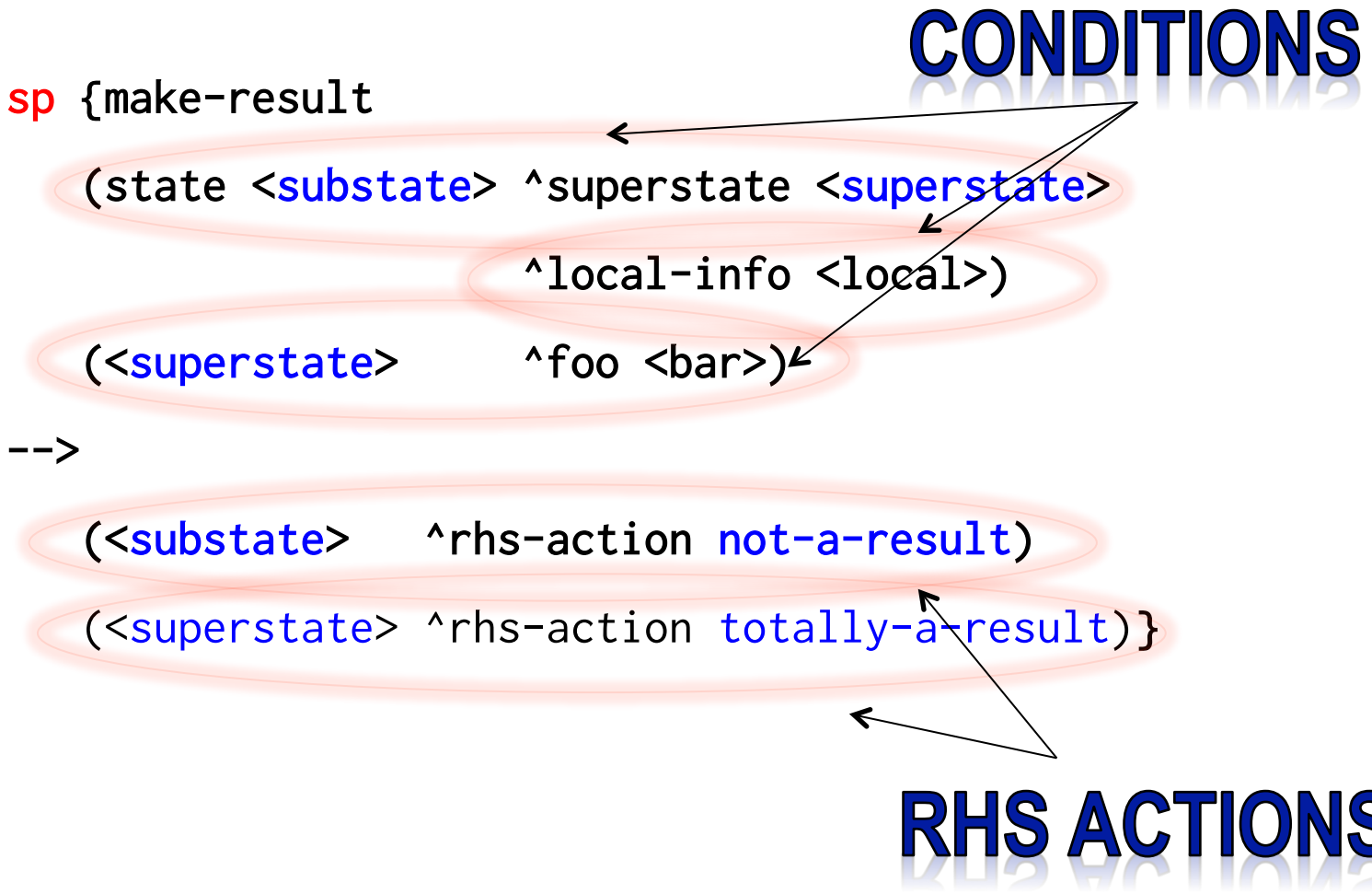
Chunking Terminology

Chunking Terminology

- Production

```
sp {make-result
   (state <substate> ^superstate <superstate>
                     ^local-info <local>)
   (<superstate>     ^foo <bar>)
-->
   (<substate>   ^rhs-action not-a-result)
   (<superstate> ^rhs-action totally-a-result)}
```

Chunking Terminology



Chunking Terminology

- Instantiation

```
(S3 ^superstate S1)
(S3 ^local-info 23)
(S1 ^foo        B1)
-->
(S3 ^rhs-action not-a-result)
(S1 ^rhs-action totally-a-result)
```

Chunking Terminology

- A *result* is working memory element that is added *to a higher level state*.

```
sp {make-result
   (state <substate> ^superstate <superstate>
                     ^local-info <local>)
   (<superstate>     ^foo <bar>)
-->
   (<substate>   ^rhs-action not-a-result)
   (<superstate> ^rhs-action totally-a-result)}
```

- This is ***when*** a chunk gets formed.

Symbol Types

- In terms of how chunking works, it's useful to delineate them into three types:
  1. Variables
     - <s>, <ss>, <o>, etc.
  2. Short-term Identifiers (STIs)
     - S1, S3, I2, etc.
  3. Non-STI's
     - Numbers: 1, 3, 1.0, 3.14
     - Long-term Identifiers (LTIs): @L1, @I2, etc.
     - Strings: Everything else

How Soar 9.3 Learns Chunks

Soar 9.3 Chunking

- Three main components:
  1. Dependency Analysis
  2. Variablization
  3. Variable Specialization

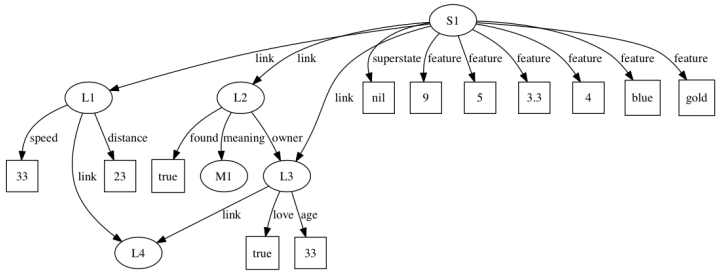
Soar 9.3 Chunking

1. **Dependency analysis**
   - Analyzes substate's problem-solving to determine necessary elements of superstate needed to produce result
   - Does this by backtracing through the working memory trace and compiling the set of all working memory elements matched that are linked to a *higher* state.

Top State

SubState

Top State



SubState

Top State

Top State WMEs

SubState

Top State

Top State WMEs

1. Agent doesn't know what to do
2. Impasse created and agent enters substate

SubState

Top State

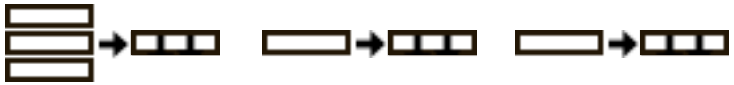
Top State WMEs

Rules fire in substate, performing problem-solving to resolve lack of knowledge, some creating substate WMEs

Substate WMEs

SubState

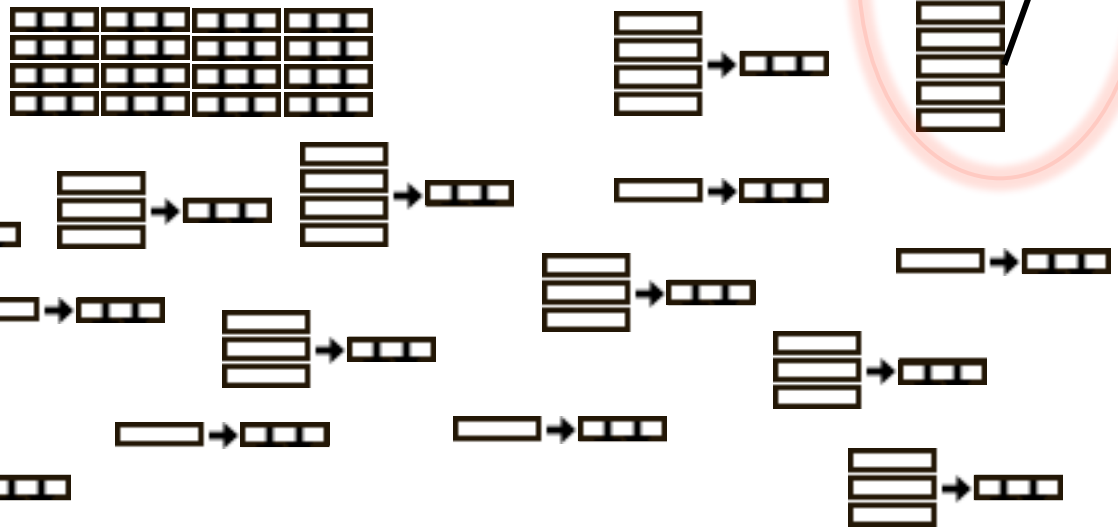
Top State



Top State WMEs

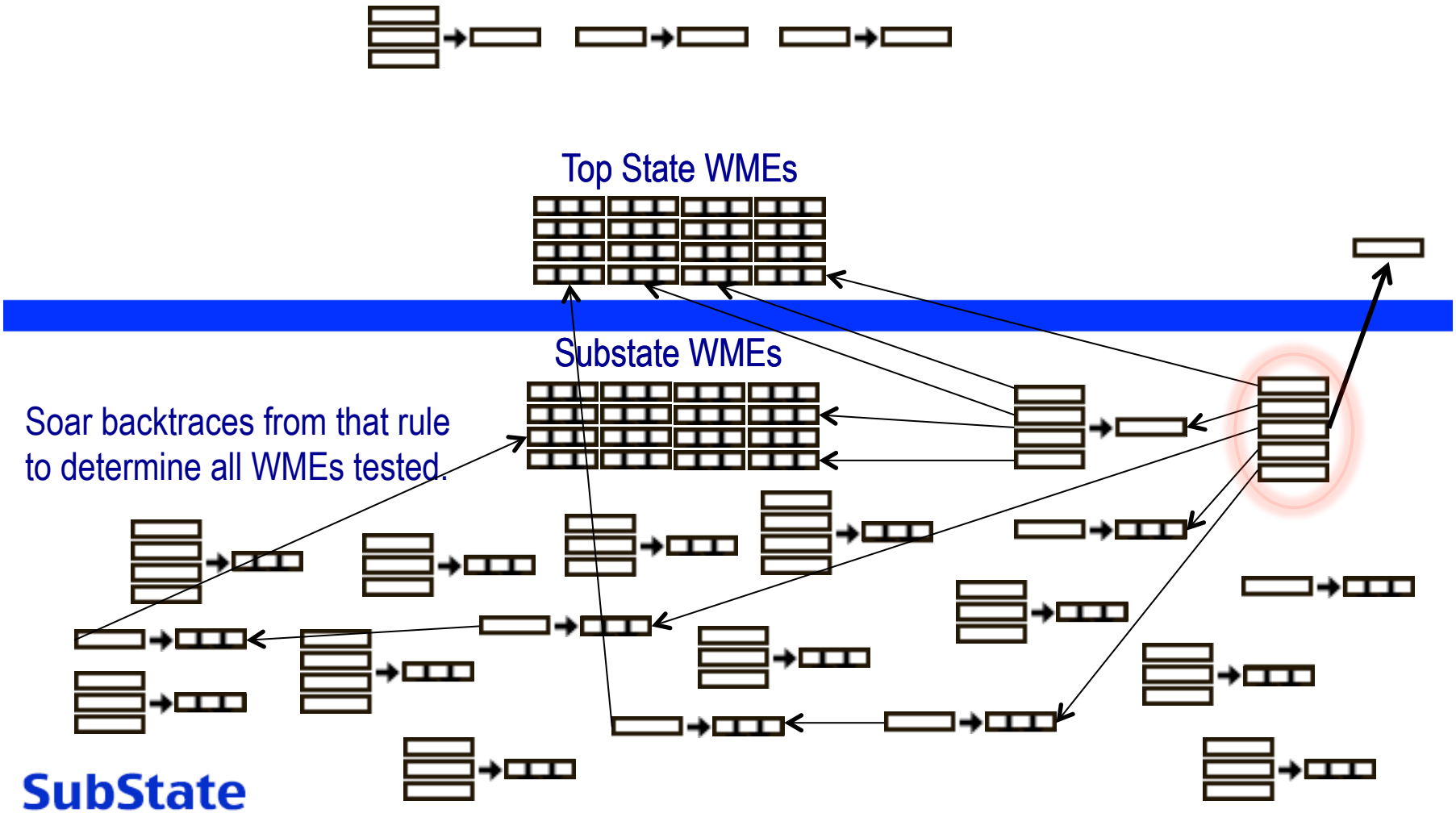
Substate WMEs

A rule fires that creates a result.

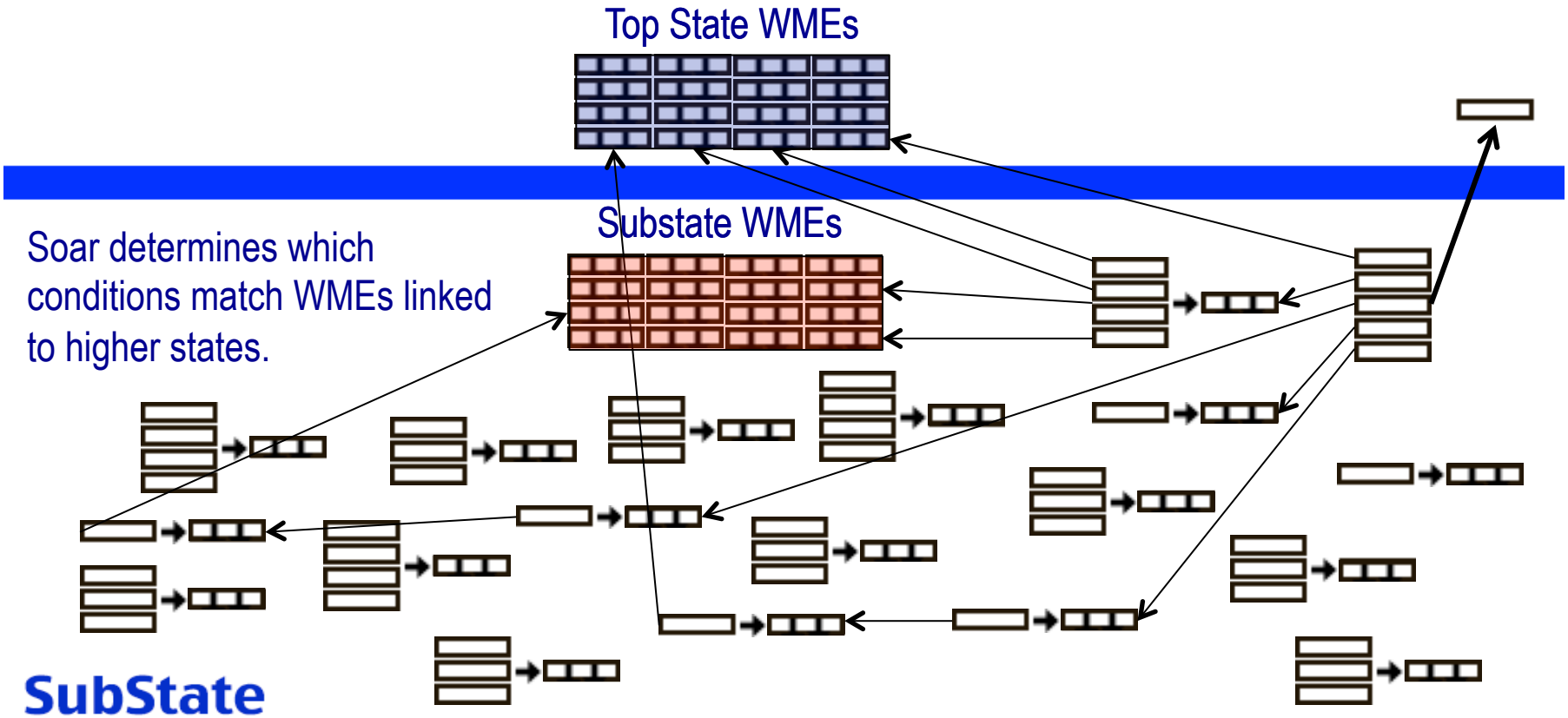


SubState

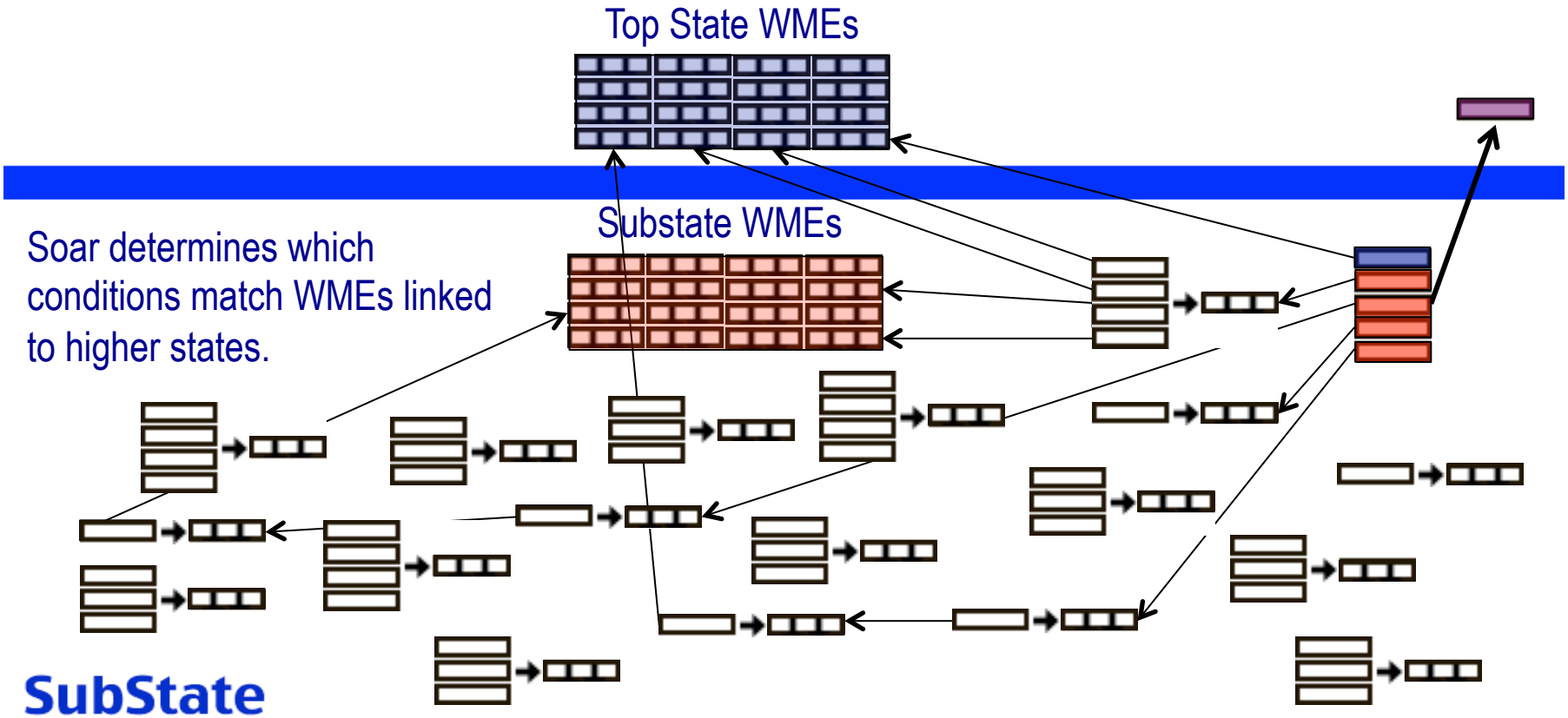
Top State


Top State WMEs
Substate WMEs
Soar backtraces from that rule to determine all WMEs tested.
SubState

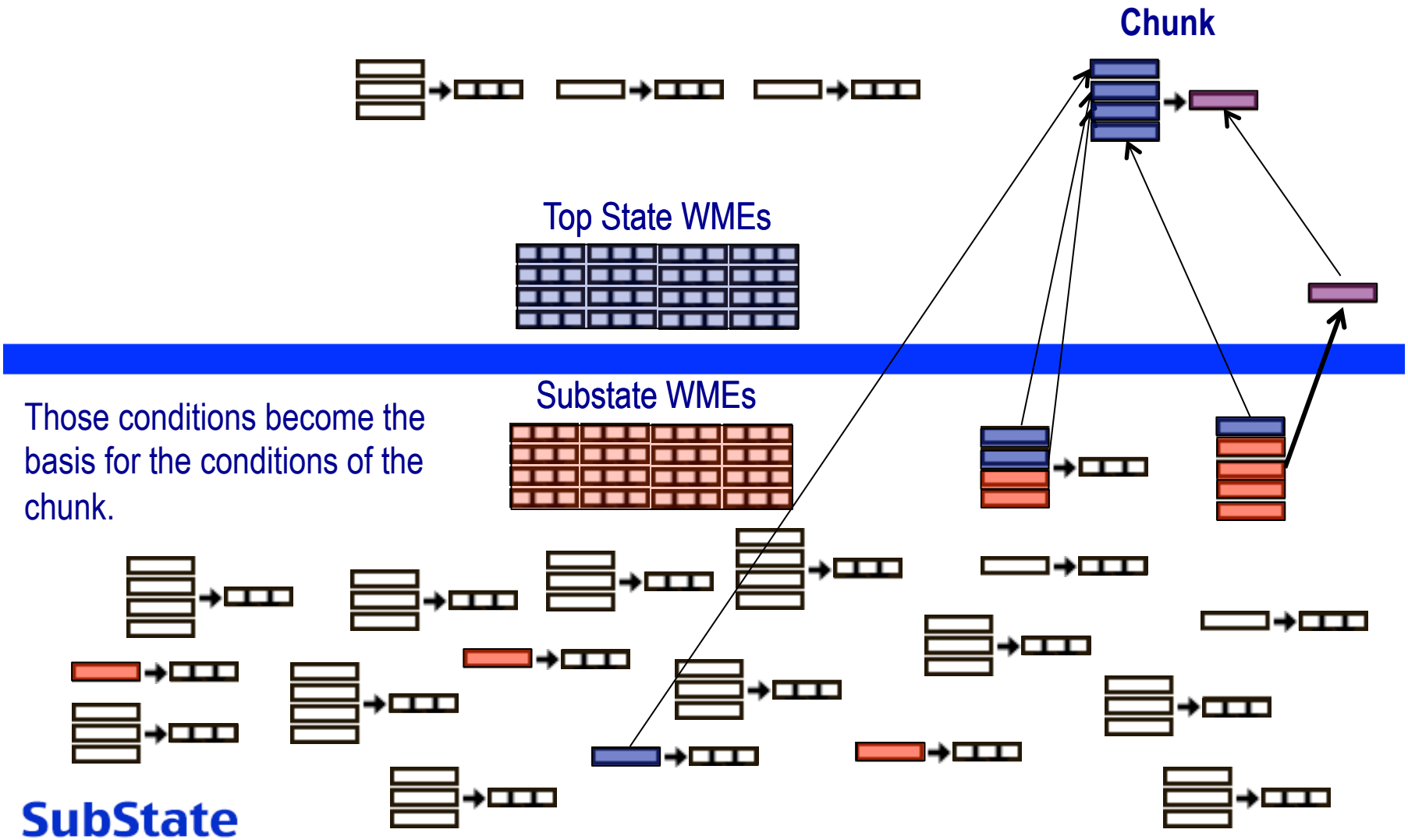
Top State


Top State WMEs
Substate WMEs
Soar determines which conditions match WMEs linked to higher states.
SubState

Top State


Top State WMEs
Substate WMEs
Soar determines which conditions match WMEs linked to higher states.
SubState

Top State


Chunk
Top State WMEs
Substate WMEs
Those conditions become the basis for the conditions of the chunk.
SubState

Soar 9.3 Chunking

2. Variablization

- Generalizes problem solving to other situations with similar relationships between STIs by substituting variables for STIs
- For example,

```
(S1   ^foo B1   ^foo B2)
```

becomes

```
(<s1> ^foo <b1> ^foo <b2>)
```

Soar 9.3 Chunking

3. Variable Specialization

- Increases specificity by possibly adding inequality constraints to variablized STIs
- For example,

```
(<s1> ^foo <b1>)
      ^foo <b2>)
```

becomes

```
(<s1> ^foo <b1>)
      ^foo { <> <b1> <b2> })
```

How Soar 9.5 Learns Chunks

Soar 9.5 Chunking

- Now has five main components:
  1. Dependency Analysis
  2. Identity Analysis
  3. Variablization
  4. Identity Unification
  5. Variable Specialization

Soar 9.5 Chunking

1. Dependency Analysis

- From the user's perspective, this aspect is essentially the same as 9.3.3.

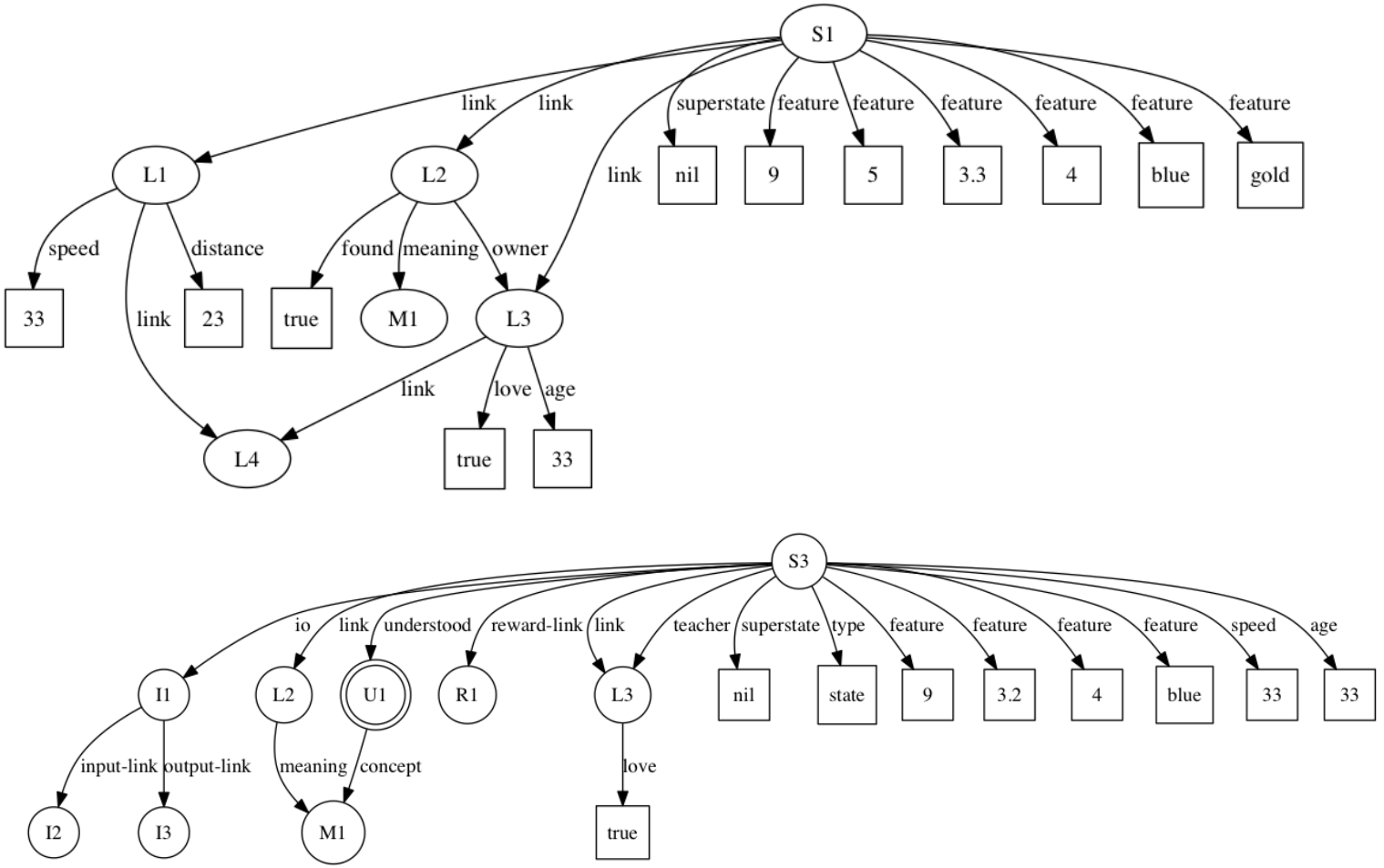
(For kernel people: Under the hood, we do a few things differently. For example, we now add all conditions linked to a higher level state, even if it matches a wme already encountered in the backtrace. This is because the new condition may have additional constraints that we need to include in the chunk.)

Soar 9.5 Chunking

2. Identity Analysis

- Analyzes grounding of symbols to determine which symbols in a substate share the same identity

Identity Analysis

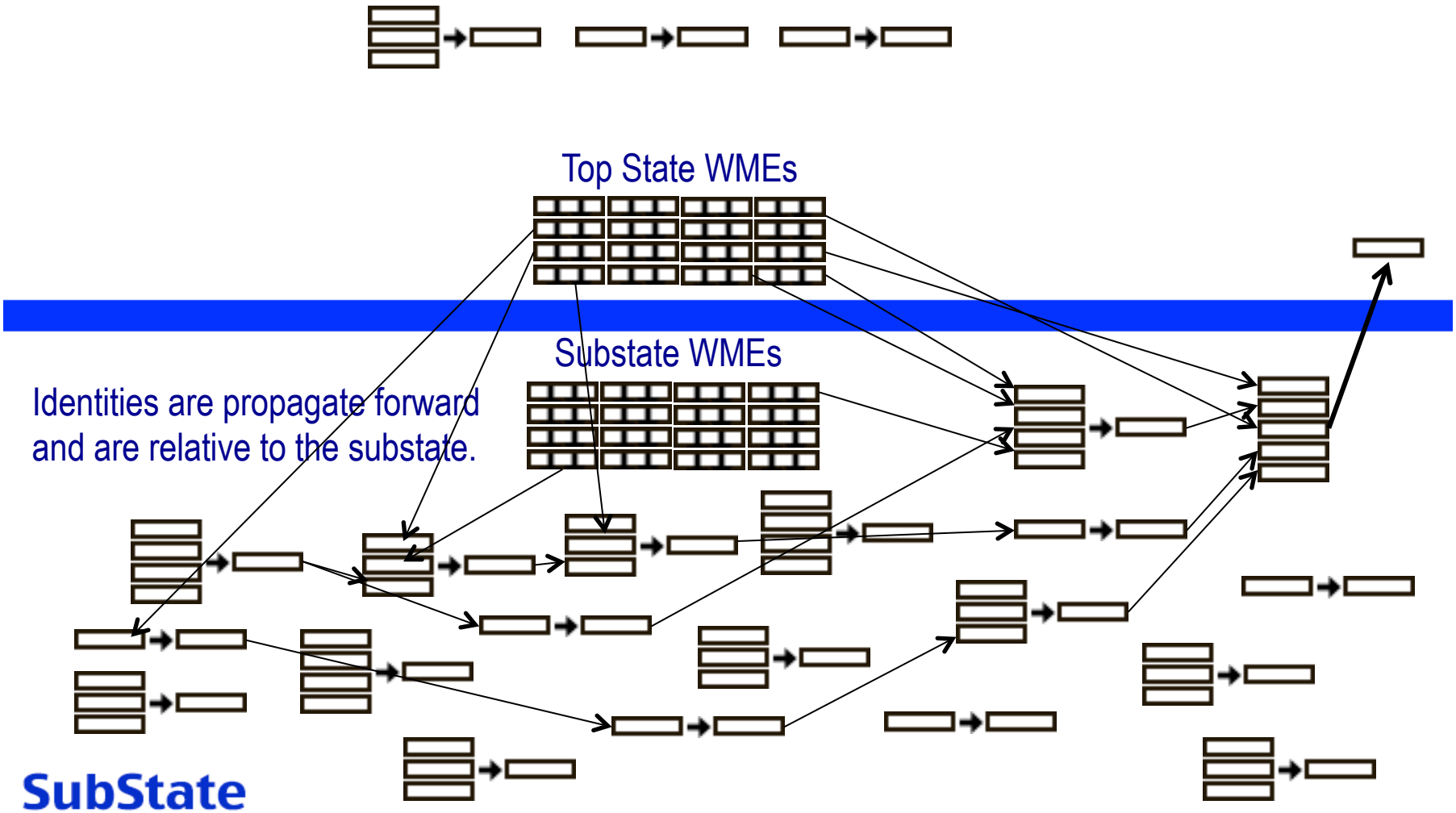


Soar 9.5 Chunking

2. Identity Analysis

- Analyzes grounding of symbols to determine which symbols in a substate share the same identity
- *Achieves this by forward propagating unique, substate-relative grounding IDs*

Top State


Top State WMEs
Substate WMEs
Identities are propagate forward
and are relative to the substate.
SubState

Soar 9.5 Chunking

4. Variablization

Differences between variablizing non-STI's and STIs

	When	With What?
STIs	Always.	Every occurrence of the same STI is replaced with the same variable.
Non-STIs	Equivalent element in matched production must be a variable	Every occurrence of a symbol with the same identity is replaced with the same variable.

Soar 9.5 Chunking

- Soar only variablizes non-STI's when equivalent element in original production is a variable

Original Production

```
sp {make-result
   (state <s> ^number 23
              ^type basketball)
→
   (<ss> ^name Jordan
         ^number 23)}
```

Chunk

```
sp {chunk-make-result
   (state <s1> ^number <n1>
               ^type basketball)
→
   (<s2> ^name Jordan
         ^number <n1>)}
```

Soar 9.5 Chunking

- Soar only variablizes non-STI's when equivalent element in original production is a variable

Original Production

```
sp {make-result
   (state <s> ^name <name>
              ^number <n>
              ^type basketball)
→
   (<ss> ^name <name>
         ^number <n>)}
```

Chunk

```
sp {chunk-make-result
   (state <s1> ^name <n1>
               ^number <n2>
               ^type basketball)
→
   (<s2> ^name <n1>
         ^number <n2>)}
```

Soar 9.5 Chunking

5. Identity Unification

- What happens if two instances of the same variable in a production matches two symbols which have the same value but different identities?

```
sp {make-result
   (state <s> ^age <favorite-number>)
              ^favorite-number <favorite-number>)
-->

   (<s> ^best-year-ever <favorite-number>)}
```

.

Soar 9.5 Chunking

5. Identity Unification

- Soar unifies the identity of the two symbols, i.e. both constants will be given the same variable in the final chunk.
- Any constant elsewhere in the same chunk that one of the two identities will also use the same variable.
- Unification is not limited to two identities.

Soar 9.5 Chunking

1. Variable Specialization

- To avoid one source of over-generality, Soar must include in the chunk any constraints on variablized constants that was required during backtracing.
- Why? We know that any matching constraints specified in the original rules are implicitly required for the problem-solving to have occurred. Otherwise, the rules wouldn't have matched in the first place.

Soar 9.5 Chunking

1. Variable Specialization

- How?
  - As Soar backtraces through the working memory trace, it collects a list of all constraints specified in the original productions.
  - When variablizing conditions, it looks for the symbols referred to in the constraints that it collected. If it finds a match, Soar variablizes that constraint and adds it to that condition in the chunk.

Nuggets

- Full desired functionality finally achieved.
- Other Soar components modified to handle new approach, for example templates and reinforcement learning.
- The wide-scale nature of the changes needed to implement this feature has allowed us to clean up and simplify many different areas of the kernel.

Coals

- This talk actually leaves out a lot of details. There are a lot of subtle aspects not discussed.
- Debugging some memory-related issues.
- Needs performance testing.
- Needs polishing.
- 9.5 has been a good demonstration of just how hard making big, low-level changes to the core aspects of Soar is with our current code base.