ACT-R as Embedded Code

Dario Salvucci
Drexel University

Specifying ACT-R Models

- At this point, we have several implementations of the ACT-R architecture
  - LISP ACT-R
  - jACT-R
  - Java ACT-R
  - Python ACT-R
  - [Distract-R]
  - [ACT-RN]
  - [Javascript ACT-R]
- Different implementations of ACT-R have taken different approaches to specifying models…

Specifying ACT-R Models

- Approach #1: Interpreted language
  - #1(a): Canonical ACT-R
  - LISP ACT-R, Java ACT-R [+ Javascript ACT-R]
  - Same model regardless of the language of the underlying interpreter
  - (Choice of language has greater effect on task implementation)

```
(sgp :esc t :lf .05)

(add-dm
    (a isa count-order first 0 second 1)
    …
    (goal isa add arg1 5 arg2 2)
)

(P initialize-addition
    =goal>
        isa         add
        …
==>
    =goal>
        sum         =num1
        count       0
    +retrieval>
        isa        count-order
        first      =num1
)
```

Specifying ACT-R Models

- Approach #1: Interpreted language
  - #1(b): Different language
  - e.g., jACT-R
  - Still, language of the underlying interpreter doesn't matter

```
<chunk name="j" type="count-order">
 <slot name="first" equals="9.0"/>
 <slot name="second" equals="10.0"/>
</chunk>

<production name="initialize-addition">
 <conditions>
  <match buffer="goal" type="add">
   <slot name="arg1" equals="=num1"/>
   <slot name="arg2" equals="=num2"/>
   <slot name="sum" equals="nil"/>
  </match>
  <query buffer="retrieval">
   <slot name="state" equals="free"/>
  </query>
 </conditions>
 <actions>
  …
 </actions>
</production>
```

Specifying ACT-R Models

- Approach #2: Embedded code

Python ACT-R

```
class MyAgent(ACTR):
    focus=Buffer()

    DMbuffer=Buffer()
    DM=Memory(DMbuffer,latency=1.0,threshold=1)

    dm_n=DMNoise(DM,noise=0.0,baseNoise=0.0)
    dm_bl=DMBaseLevel(DM,decay=0.5,limit=None)

    def init():
        DM.add('customer:customer1 condiment:mustard')
        focus.set('rehearse')

    def request_chunk(focus='rehearse'):
        print "recalling the order"
        DM.request('customer:customer1 condiment:?condiment')
        focus.set('recall')
```

Specifying ACT-R Models

- Approach #2: Embedded code

Distract-R

```
// control-attend-near
if ((na == NILVAL) && (when == NILVAL)
                  && model.getVision().isVisionFree()
                  && model.getVision().getVisualLocation() == null
                  && model.getVision().getVisual() == null) {
          model.trace("DRIVE", "control-attend-near");
          na = NONEVAL;
          model.getVision().startVisualLocation(Chunk.KIND_NEAR);
          return true;
}

// control-attend-near-wait
if ((na == NILVAL) && (when != NILVAL) && (when <= model.getTime())
                  && model.getVision().isVisionFree()
                  && model.getVision().getVisualLocation() == null
                  && model.getVision().getVisual() == null) {
          model.trace("DRIVE", "control-attend-near-wait");
          na = NONEVAL;
          model.getVision().startVisualLocation(Chunk.KIND_NEAR);
          return true;
}
```

Specifying ACT-R Models

- So, which approach is best?
- Each approach apparently has its value — why else would people have made them :)
- But it's useful to think about some issues…

Issue #1: Constraints

- A modeling language constrains the user to specify knowledge/behavior in a very particular way
  - one of the hallmarks of a cognitive architecture
  - a unified approach to knowledge representation
- But often, rules are bent/broken to address components outside the model's scope
  - e.g., the dreaded !eval!
  - not necessarily a bad thing
    - might just be a way to abstract over things beyond the model
  - if you're concerned about building useful models, it can be used in a productive way

Issue #1: Constraints

- Two examples…
  - Chunks vs. equations
    - e.g., the driver model uses an equation to compute steering angle from visual points
      - this is really a stand-in for retrieval of chunks, learned over time… but abstracts over this issue for simplicity
  - Dynamic chunks
    - e.g., large-scale database of declarative chunks likely needs to be implemented differently
      - perhaps, create chunks on the fly, rather than storing all
      - (analogous problem to equations)
    - e.g., natural language
      - what if we wanted to store parts of speech in ACT-R? how might this be implemented?

Issue #2: Procedural Learning

- For an interpreted model, rules are created at the start, but can be changed on the fly
  - a la production compilation
- Embedded code doesn't (easily) allow for procedural learning
- Embedded code also encourages a sequential style of behavior description — not as rules evaluated in parallel
  - in my mind, it seems to be an open question of how many models gain from this flexibility

Issue #3: Model Integration

- In theory, a modeling language facilitates integration of 2+ models
  - they're all written in the same language, using the same cognitive representations
- In practice, as we know, this doesn't happen much
  - the "API" between models is difficult to validate
  - embedded code helps to enforce the API
    - because of type checking, including packages/libraries, etc.
    - e.g., by defining types and specific slots
- Again, it comes down to what's easy & useful
- Which brings us to our user base…

Potential Users

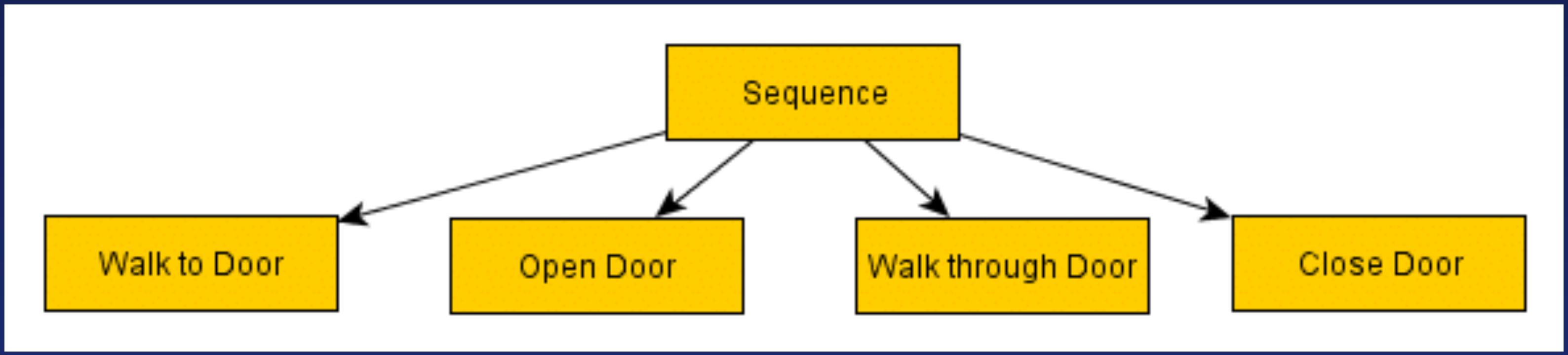
- We've largely targeted ACT-R to other cognitive scientists
  - they are trying to understand cognition
  - they care, first and foremost, about the model
  - user base: maybe 100-1000 people
- Meanwhile, there are plenty of "agent builders" interested in coding behavioral models
  - e.g., "behavior trees" for gaming
  - the "model" isn't the 1st, or 7th, thing on their mind
  - they need something that integrates quickly and easily
  - potential user base: >>1000 people

"Agent Builder" Needs

- Get up and running quickly?
  - download a library, get code from a tutorial, integrate
  - interpreted ACT-R?
    - right now, fairly difficult, especially the glue between the task and what the model sees
  - embedded code?
    - potentially much faster — if the programming language matches
- Language interoperability?
  - they can't conform to our language (they already have 100k lines of code in another language)
  - (we might spend lots of time integrating a Unity game with LISP code, but I doubt anyone else would)

"Agent Builder" Needs

- Access behavior at different levels of abstraction?
  - do they need an actual running model?
  - or are they looking for smaller functions??
    - e.g., calculate mouse movement or keystroke time
    - e.g., calculate response time for a visual search
- Visual editors and IDEs?
  - game behavior-tree designers rely heavily on these…



Prototype System

```
class ClickWorld extends World {
     private Display display;
     private Item button;

     ClickWorld() {
          super();
          display = new Display();
          button = new Item("button", 0, 0, 30, 30);
          button.addClickListener(new ClickListener() {
               @Override
               public void click() {
                    moveButton();
               }
          });
          display.add(button, "X");
          moveButton();
     }
     …

     void moveButton() {
          display.move(button, 50 + random.nextInt(200), 50 + random.nextInt(200));
          log("move");
     }

     public static void main(String args[]) {
          ClickWorld world = new ClickWorld();
          new Simulation(new ClickAgent(world)).setRealTime(true).run();
     }
}
```

Prototype System

```
class ClickAgent extends Agent {
    private ClickWorld world;
    private DesktopVision vision;
    private DesktopMotor motor;

    ClickAgent(ClickWorld world) {
        super();
        this.world = wo
        Display display
        vision = new Deskto                        ));
        motor = new DesktopMoto  is, display, vision);
    }

    @Override
    public void run() {
        while (!world.isDone()) {
            vision.waitFor(new Pattern(Item.      "button"));
            motor.pointAndClick(vision.getFound());
        }
    }
}
```

Start the visual request

Block this thread until visual item is found

Prototype System

```
public class Numbers implements MemoryModule {
    …

    public Numbers() { … }

    public NumberChunk get(int n) { … }

    @Override
    public void addStaticChunks(Memory memory) {
    }

    @Override
    public MemoryChunk getDynamicChunk(Pattern pattern) {
        NumberChunk chunk = null;
        if (pattern.has("isa", Operator.EQ, "number")) {
            SlotPattern slotPattern = pattern.get("value", Operator.EQ);
            if (slotPattern != null) {
                Integer value = (Integer) slotPattern.getValue();
                if (value != null)
                        chunk = get(value);
            }
        }
        return chunk;
    }
}
```

Prototype System

```
public class Counting extends Module {
    private Memory memory;
    private Speech speech;

    public Counting(Agent agent, Memory memory, Numbers numbers, Speech speech) {
        super("counting", agent);
        this.memory = memory;
        this.speech = speech;
        memory.include(numbers);
    }

    public void count(int from, int t
        while (from <= to) {
            memory.recall(new Pattern("isa
                            .add(NumberC    k.VALUE, from));
            speech.say(memory.getRecalled().getString("name"));
            from = memory.getRecalled().getInteger("next");
        }
    }
}
```

Block this thread until item is recalled

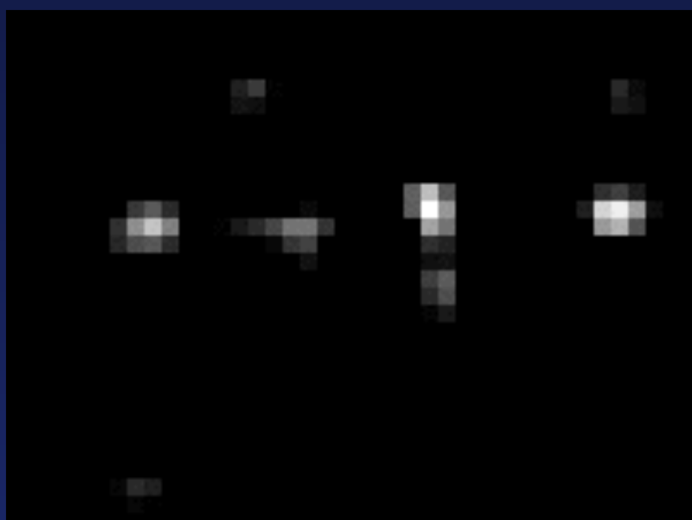
Thinking Ahead

- The prototype system is still built for simulating and acting, not other levels of abstraction
- Let's look at some examples…
  - Visual Search
  - Arithmetic
  - List Memory

Thinking Ahead

- Visual Search
  - example: iLab Vision C++ Toolkit (Itti et al., USC)
  - takes raw image as input, can generate as output…

saliency map

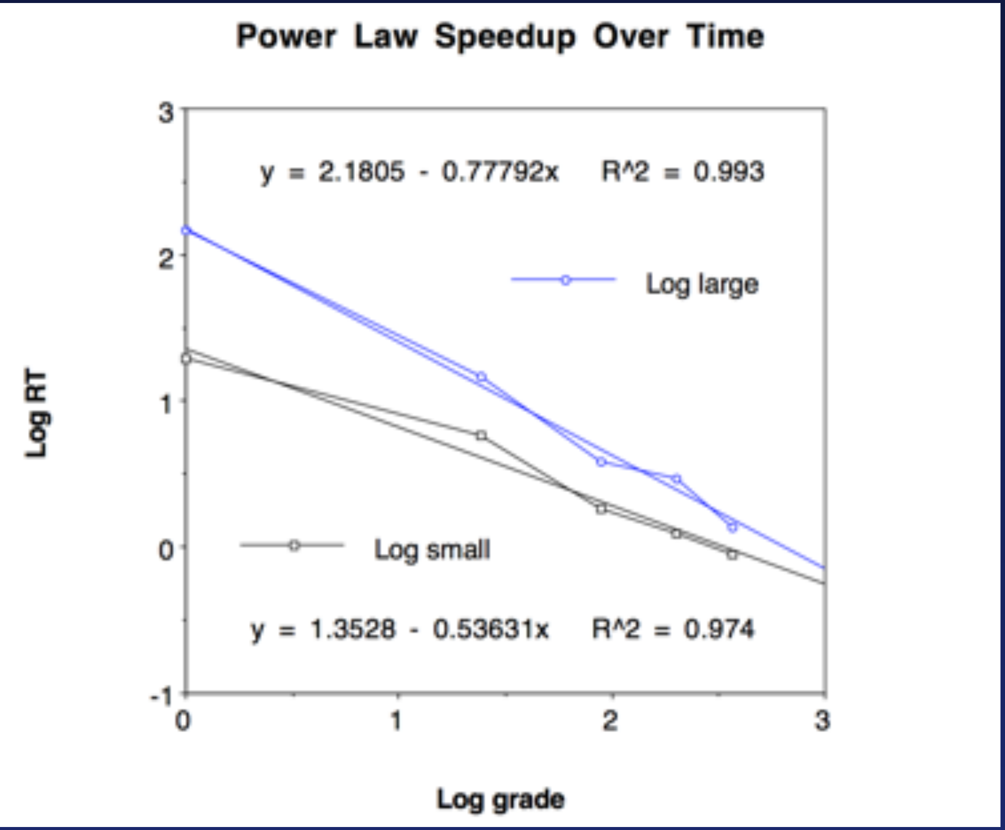


shifts of attention based on saliency

  - strong predictions, easy to integrate with other models

Thinking Ahead

- Arithmetic (Lebiere, 1998)
  - RT, small & large problems over time…



Thinking Ahead

- Arithmetic (Lebiere, 1998)
  - % correct vs. incorrect responses (age 4)

	0	1	2	3	4	5	6	7	8	9	10	11	other
1+1	0	5	86	0	2	0	2	0	0	0	0	2	4
1+2	0	0	9	70	2	0	4	0	0	7	2	2	5
1+3	0	2	0	11	71	5	2	2	0	0	0	0	7
1+4	0	0	0	0	11	61	9	7	0	0	0	2	11
1+5	0	0	0	0	13	16	50	11	0	2	2	0	5
2+1	0	7	5	79	5	0	0	0	0	0	0	0	4
2+2	2	0	4	5	80	4	0	5	0	0	0	0	0
2+3	0	0	4	7	38	34	9	2	2	2	0	0	4
2+4	0	2	0	7	2	43	29	7	7	0	0	0	4
2+5	0	2	0	5	2	16	43	13	0	0	2	0	18
3+1	0	2	0	9	79	4	0	4	0	0	0	0	4
3+2	0	0	9	11	11	55	7	0	0	0	0	0	7
3+3	4	0	0	5	21	9	48	0	2	2	2	0	7
3+4	0	0	0	5	11	23	14	29	2	0	0	0	16
3+5	0	0	0	7	0	13	23	14	18	0	5	0	20
4+1	0	0	4	2	9	68	2	2	7	0	0	0	7
4+2	0	0	7	9	0	20	36	13	7	0	2	0	7
4+3	0	0	0	5	18	9	9	38	9	0	2	0	11
4+4	4	0	0	2	2	29	7	7	34	0	4	0	13
4+5	0	0	0	0	4	9	16	9	11	18	11	4	20
5+1	0	0	4	0	4	7	71	4	4	0	4	0	4
5+2	0	0	5	20	2	18	27	25	2	0	2	0	0
5+3	0	0	2	11	9	18	5	16	23	0	5	0	11
5+4	0	0	0	0	11	21	16	5	11	16	4	0	16
5+5	4	0	0	0	0	7	25	11	2	4	34	4	11

Thinking Ahead

- Arithmetic (Lebiere, 1998)
  - as a library? runnable actions, simpler functions…

```
public class Arithmetic {

    public Arithmetic(int age) { … }

    public int add(int x, int y) {
        // performs addition with RT, correctness
    }

    public double getProbabilityCorrect(int x, int y) {
        // returns probability correct for this age
    }

    public double getResponseTime(int x, int y) {
        // returns predicted RT for this age, with noise
    }

    …
}
```

Thinking Ahead

- List Memory (e.g., Anderson, Bothell, Lebiere, Matessa, 1998)
  - as a library? …

```
public class ListMemory {

    public ListMemory() { … }

    public void clear() { … }

    public void add(String word) { … } // assumes running time
    public void add(String word, double t) { … } // specifies time

    public List<Recalled> recall(double t) {
        // given current time, returns recalled list with RT and errors
    }

    …
}
```

Summary

- Hunch: Embedded code will facilitate integration and sharing — at least for "agent builders" — in a way that interpreted language doesn't
  - at least, for domains where production learning isn't necessary or critical
- Only sketches of a prototype system at this point
- The proof would come in implementations of sample domains, like arithmetic, list memory, etc.
- … and their use by actual users