CAP6938-02
Plan, Activity, and Intent Recognition

Lecture 3:

Event Hierarchy Circumscription (cont);
Event Tracking in SOAR

Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu
Schedule: T & Th 1:30-2:45pm
Location: CL1 212
Office Hours (HEC 232):
T 3-4:30pm, Th 10-11:30am

Outline

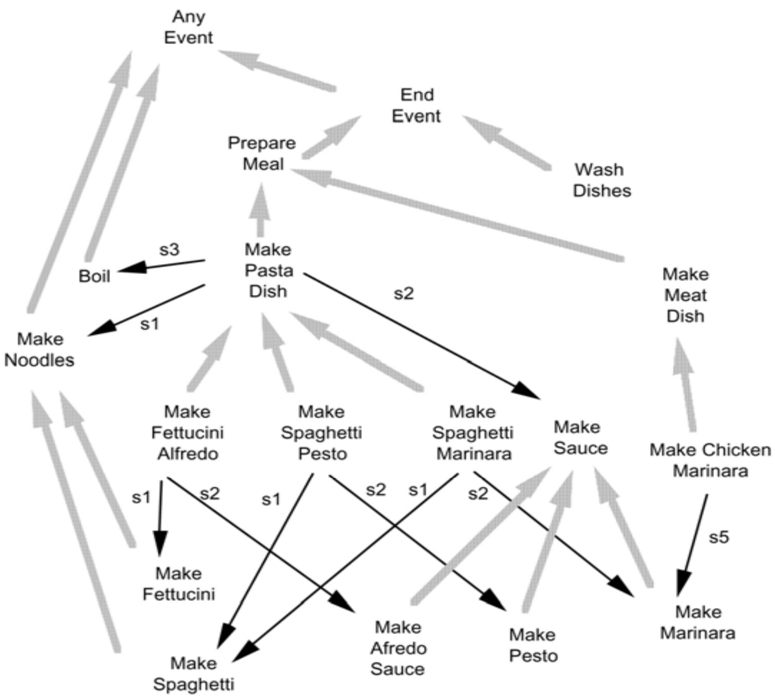
- Finish discussing Kautz paper (relevant pages pp. 1-24, pp 46-47, pp 63-65)
- Student presentations of final research project
- New domain: Opponent modeling for games and battlefield analysis
- Background on SOAR/Tac-Air SOAR
- Event tracking in SOAR

Kautz's Model

- First order predicate calculus
- Event hierarchy (logical encoding of a semantic network)
  - Event predicates
  - Abstaction axioms
  - Decomposition axioms
- General axioms: hardest to use for inference
  - Includes temporal constraints between the steps
  - Equality constraints between the agents executing steps or objects involved in steps
  - Preconditions
- Special event predicates*: End, AnyEvent (*top-level abstraction*)*

Event Hierarchy Circumscription

Event hierarchy



General axioms

$\forall x$. *MakePastaDish(x)* $\supset$

	MakeNoodles(step1(x)) $\wedge$
Components	*MakeSauce(step2(x))* $\wedge$
	Boil(step3(x)) $\wedge$
Equality	*agent(step1(x))* = *agent(x)* $\wedge$
Constraints	*result(step1(x))* = *input(step3(x))* $\wedge$
Temporal	*During(time(step1(x)), time(x))* $\wedge$
Constraints	*BeforeMeets(time(step1(x)), time(step3(x)))* $\wedge$
	Overlaps(time(x), postTime(x)) $\wedge$
Preconditions	*InKitchen(agent(x), time(x))* $\wedge$
	Dexterous(agent(x)) $\wedge$
Effects	*ReadyToEat(result(x), postTime(x))* $\wedge$
	PastaDish(result(x))

H. Kautz, A Formal Theory of Plan Recognition and its Implementation, in <u>Reasoning about Plans</u>

Closed-world Assumptions

- What are they?
- Exhaustiveness
- Disjointedness
- Component/use assumptions
- Minimum cardinality assumptions
- Are observations assumed to be complete?

Exhaustiveness

- Known ways of specializing an event type are the only ways of specializing it
- Example: the only pasta dishes which exist are {fettucini alfredo, spaghetti pesto, spaghetti marinara}
- Allows Sherlock Holmes style conclusions:
  - Not fettucini alfredo
  - Not spaghetti pesto
  - Must be spaghetti marinara!

Disjointedness

- Types are disjoint, unless one abstracts the other, or they abstract a common type
- Can't invent new dishes "meat ravioli" that abstract both the meat dish and the pasta dish
- Similar to exhaustiveness but for event types
- Allows the conclusions to be made:
  - Making a pasta dish
  - Therefore agent is not making a meat dish (since neither abstracts each other)

Component/Use Assumption

- Seeing an event implies the disjunction of the plans which include it as a component
- Agent is boiling water...
  - must be a pasta dish since nothing else includes that event.
- Allows for missing but not erroneous observations

Minimum Cardinality Assumption

- Assume parsimony: the minimum number of plans to explain the observations
- Without this assumption each event could always belong to a separate plan
- If you observe the event “get gun” and “go to woods”, assume that both are related to the plan “hunt” rather than believing that the person is going hunting and robbing a bank

Kautz’s Inference Procedure

Note that there have been faster, more specialized inference procedures developed to handle Kautz’s model

- From observations, apply component use assumption and abstraction axioms to find all top-level plans contain the observed event.
- Apply other constraints expressed by general axioms locally (this is where most of the work occurs)
- Combine information from multiple observations using the minimum cardinality assumption to minimize the number of plans under consideration

Summary

- Handles well:
  - Incomplete sequences of observations
  - Plans that lack total ordering
- Handles poorly:
  - Errors in observations
  - Situations with large numbers of possible, but improbable, plans
- In contrast, probabilistic frameworks handle those cases quite well...

Student Project Presentations

Domain: Opponent Modeling

- How does plan recognition differ in adversarial domains than in non-adversarial ones?
- More time pressure
- Smart opponents will deliberately mislead you.
- Performance is usually measured by an improvement in agent's planning rather than recognition accuracy
- Game-theoretic methods work well: assume the opponent is strengthening its position and harming you

SOAR

- Stands for **S**tate, **O**perator, **A**nd **R**esult
- URL: http://sitemaker.umich.edu/soar/home
- Developed from Newell and Simon’s General Problem Solver (GPS)
- Original purpose: to create a cognitive architecture that could integrate both goal-driven and reactive behavior
- Now: mainly used as a planning/execution system for simulated agents (especially in military simulation applications)
- What’s the difference between cognitive architecture and any other type of planning system?

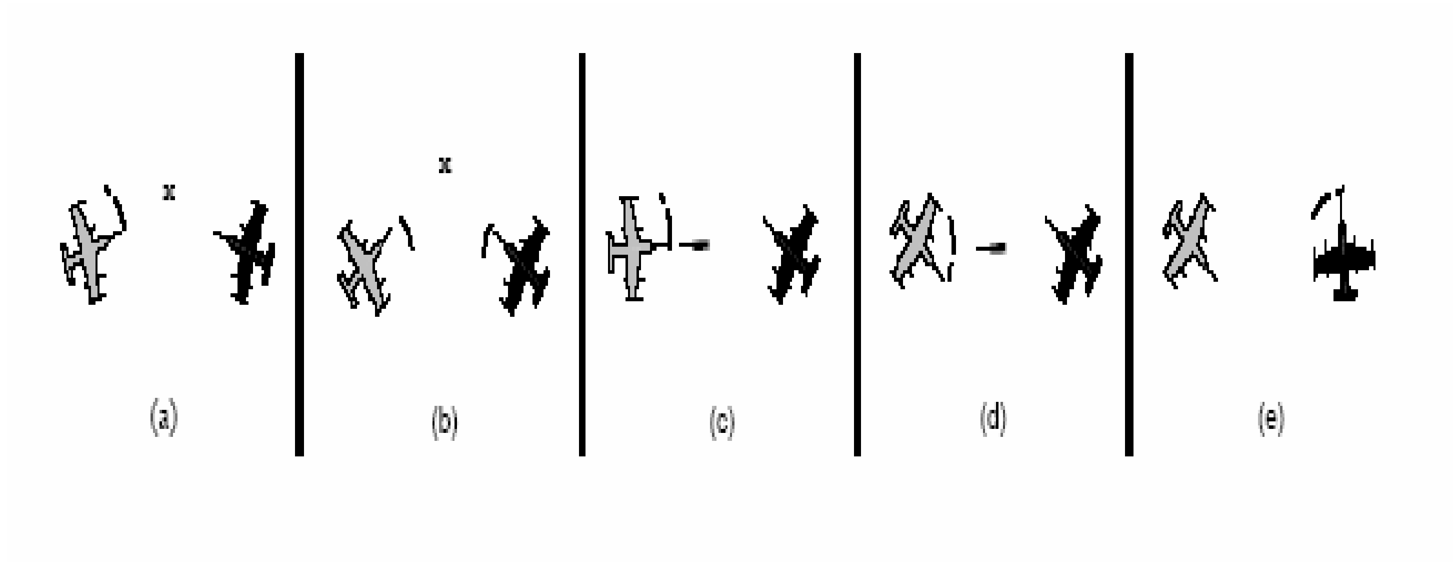
SOAR Definitions

- Problem space: set of states (situations) plus set of operators (actions)
- System cycles through proposing, selecting and applying operators
- Knowledge encoded as productions (condition-action rules)
- All relevant productions trigger in parallel whenever changes in goals, state, and perception cause conditions to be met.
- Impasses solved through subgoaling, solution remembered by chunking.

Tac-Air SOAR Testbed

- SOAR plus set of perceptual and motor interfaces that allow agents to pilot aircraft in DIS (Distributed Interactive Simulation)
- Focus is on beyond-visual-range combat where pilots rely on radar and communication
- Demonstrated in Simulated Theater of War (STOW-97):
  - Mission types: defensive counter air, close air support, suppression of enemy air defense, strategic attack, escorts, tankers, airborne early warning and reconn/intel.
  - Demonstrated that 2 people could monitor 70 simultaneously active agents

Recognize Flight Manuevers



Observations: enemy flies towards you, then turns to a certain angle

Want to recognize: enemy fired an (unseen missile), then did an FPOLE maneuver

Agent should execute: missile evasion maneuver

Problem Characteristics

- Events are not the result of a single agent's actions
- Agent must consider the actions of multiple agents simultaneously
- Agent must consider the effect of its own actions.
- Real-time, continuous vs. one shot recognition
- Ambiguity in the opponent's behaviors

Solution

- Simple insight: model what you would do if you were in the opponent's position
- What are problems with this?
  - High overhead: must program an agent capable of solving the problem
  - Modeling the opponent's world state can be difficult (what is the opponent's sensor model?)
  - Maintaining multiple hypotheses is even more expensive
- What are the strengths?
  - Allows designer to leverage extra domain knowledge
  - Does not require enumerating chains of possible events

Ambiguity in Event Tracking

- Ambiguity: the bane of plan recognition!
- Potential solutions:
  - Maintain multiple operator hierarchies (continue considering all valid hypotheses)
  - Delay until more evidence presents itself
- Tambe solution: attempt to resolve ambiguity and commit to a single interpretation
  - Passive ambiguity resolution (game-theoretic)
  - Active resolution: modify agent's actions to resolve ambiguity
  - Detect incorrect interpretation through match failure
  - Recovery mechanisms (assumption injection, backtracking)

Other Issues

- Spending time on recognition vs. computation
- Feature selection: which details should the agent pay attention to?
- SOAR-specific issues with maintaining multiple problem spaces (world-centered problem space)
- Incomplete plan libraries
- What category does this type of plan recognition system fall under?
- How did Tambe evaluate this system?

Next Time

- Conclusion of student presentations
- Application area: monitoring teammates's actions
- Efficiency improvements for symbolic plan recognition (Kaminka paper)