Task Supervision Using Formal Languages

Supervising Tasks Based on Few Expert Examples

VRAIN
Valencian Research Institute for Artificial Intelligence

GENERALITAT VALENCIANA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

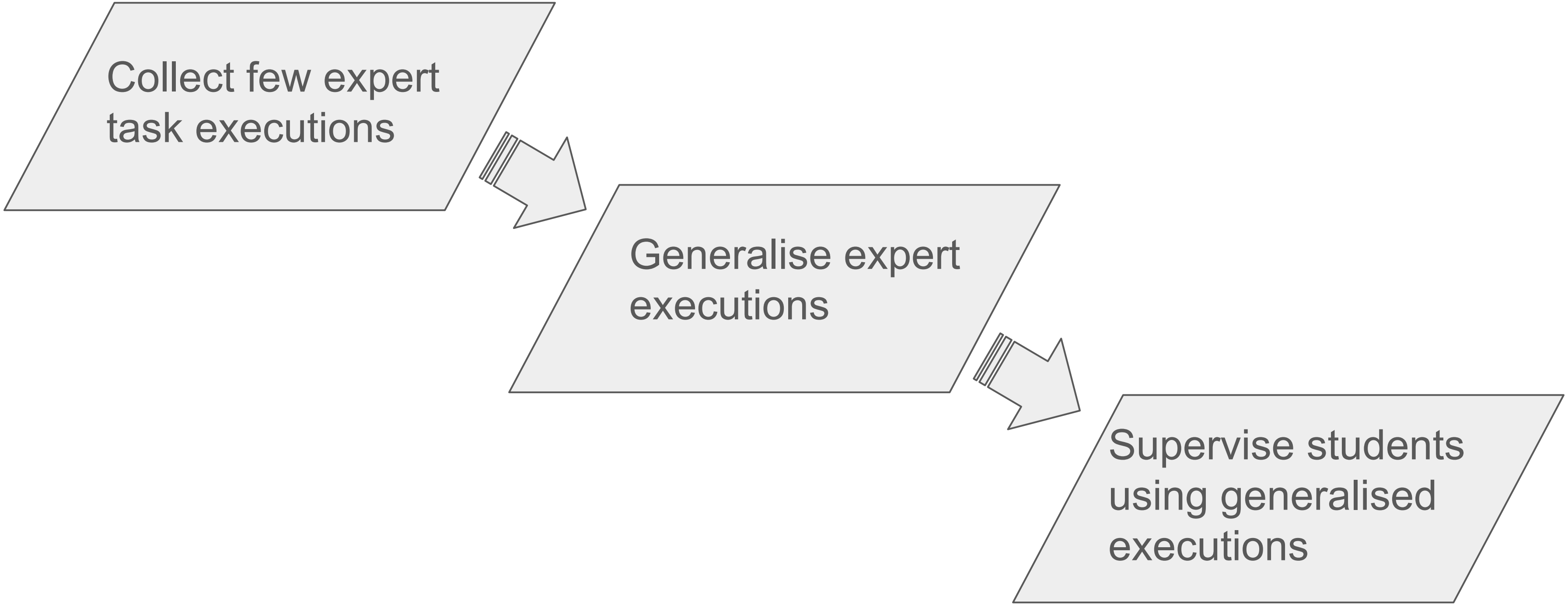
Machine Teaching Humans

– Teaching involves expert supervision of task executions by students
    – Mistake recognition
    – Evaluation
    – Correction

– Teaching is time-consuming, expensive and doesn’t scale well

– Machine supervision is more optimal than human supervision
    – Scaling
    – Cost-effective

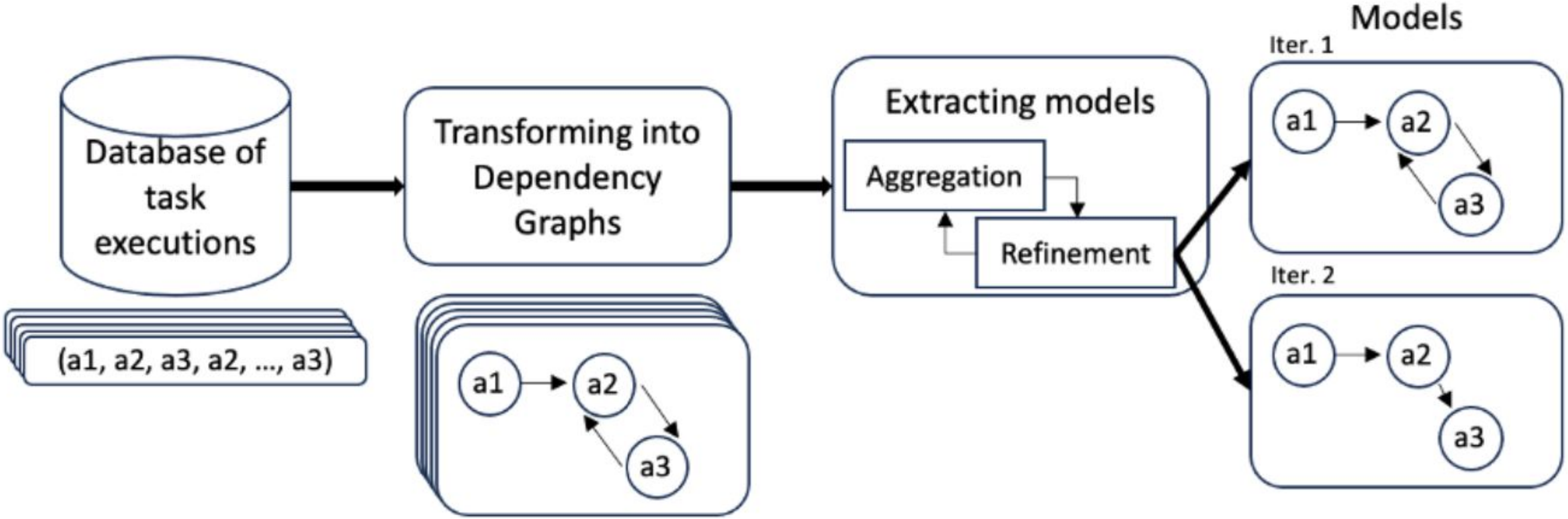
Generalised Approach



The Difficulties of Task Generalisation

- Generalising tasks based on event sequences can be challenging
- Generalising the task of making a salad is a great example:
  - Many different recipes for a salad
  - Different sequences and ingredient sets
- Generalising all recipes can lead to unexpected results

Generalising Expert Executions – Based on Prior Work



Nieves, D., Ramírez-Quintana, M., Monserrat, C., Ferri, C., Hernández-Orallo, J.: "Learning alternative ways of performing a task." Expert Systems with Applications 148, 113263 (2020). https://doi.org/10.1016/j.eswa.2020.113263

Advantages and Disadvantages of Formal Methods for Task Supervision

– Expressiveness

– Formal reasoning

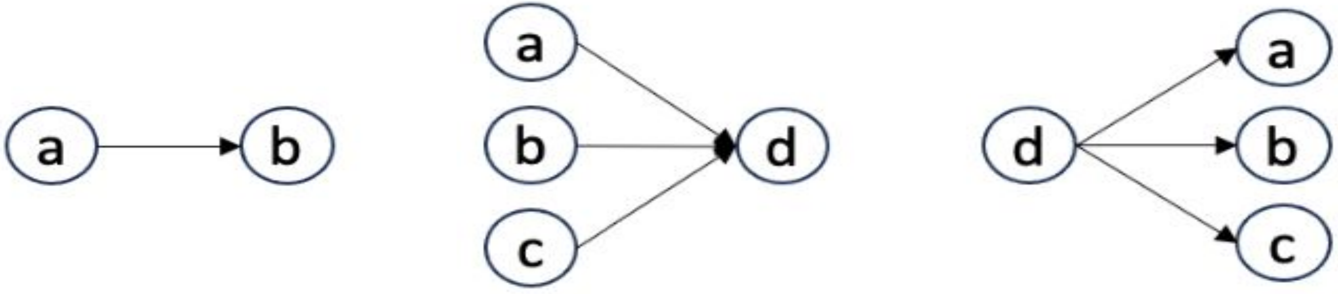
– Efficient at handling tasks and their representations

Event Calculus – First Order Logic Language for Events and their Effects

Predicates:	
`initiates(E,F,T)`	Event `E` initiates (makes true) the fluent `F` from time `T+1`.
`terminates(E,F,T)`	Event `E` terminates (makes false) the fluent `F` from time `T+1`.
`holdsAt(F,T)`	Fluent `F` is true at time `T`.
`stoppedIn(`T_1`,F,`T_2`)`	Fluent `F` is terminated in an instant of time between T_1 and T_2.
`happens(E,T)`	Event `E` occurs at time `T`.
General Axioms from EC:	
`initiates(E, started(E), T)`	`:- happens(E,T).`
`terminates(E, started(`E_1`), T)`	`:- happens(E,T), holdsAt(started(`E_1`),T).`
`initiates(E, completed(`E_1`), T)`	`:- happens(E,T), holdsAt(started(`E_1`),T).`
`holdsAt(F,T)`	`:- happens(E,`T_1`), initiates(E,F,`T_1`),` `not stoppedIn(`T_1`,F,T),` $T_1 < T$.
`stoppedIn(`T_1`,F,`T_2`)`	`:- happens(E,T),` $T_1 < T$, $T < T_2$, `terminates(E,F,T).`

Our Approach using First Order Logic Languages

– Encoded dependency graph divided into three parts

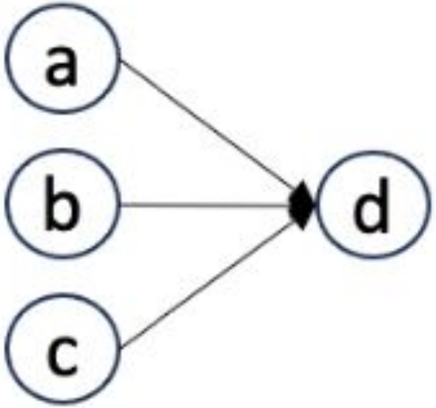


Event Calculus Encoding

Encoded as:

: −happens(b, T), not predecesor(a, b, T).

Event Calculus Encoding



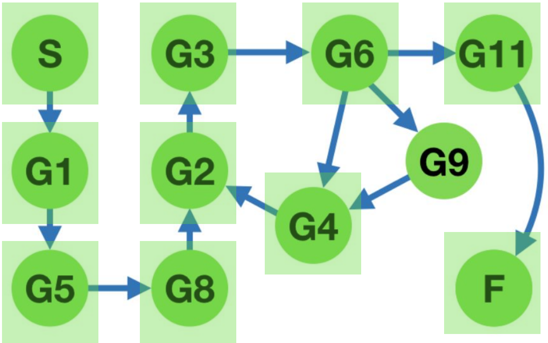
CASE	TRANSLATION
AND	$:-\mathtt{happens}(d,T), \mathtt{happens}(a,T_1), T_1 < T, \mathtt{not\ bbetween}(a,d,T),$ $\mathtt{happens}(b,T_2), T_2 < T, \mathtt{not\ bbetween}(b,d,T),$ $\mathtt{happens}(c,T_3), T_3 < T, \mathtt{not\ bbetween}(c,d,T).$
OR	$:-\mathtt{happens}(d,T), \mathtt{not}\ 1\{\mathtt{predecesor}(a,d,T), \mathtt{predecesor}(b,d,T),$ $\mathtt{predecesor}(c,d,T)\}.$
XOR	$:-\mathtt{happens}(d,T), \mathtt{previous}(d,T,T_1), \mathtt{not}\ 1\{\mathtt{happens}(a,T_2),$ $\mathtt{happens}(b,T_2), \mathtt{happens}(c,T_2) : T_1 < T_2 < T\}1.$

Our Approach using Clingo

– Encoded Sequence of Events

```
happens(gS,1).
happens(g1,2).
happens(g5,3).
happens(g8,4).
happens(g2,5).
happens(g3,6).
happens(g6,7).
```

– Sequence Evaluation



S, G1, G5, G8, G2, G3, G8, G2, G3, G2, G8, G2, G3, G6, G4, G2, G3, G6, G4, G2, G3, G8, G2, G3, G6, G4, G2, G3, G6, G11, F

Alternative Approach with Maude

– Powerful declarative language
– Rewriting logic
– Execution analysis (XAI)
– Counterexamples (XAI)
– UPV involvement in development

Future Work

– Adjusted approach towards task generalisation

– Use of expert description of the task to check and complement supervision (Maude)

– Adding more potential to the supervision process (and split, or split, xor split, combinations, …).

– Model enhancement with expert knowledge