Knowledge-based programs as plans

Jérôme Lang (LAMSADE, Paris) & Bruno Zanuttini (GREYC, Caen)

ECAI-2012 + TARK-2013 + IJCAI-2015
+ ongoing work (with Anaëlle Wilczinski, LAMSADE)
A card game program

Goal:
- pick some cards, maximum 5
- try to obtain three cards of the same rank

Do
- pick a card $c$
- look at the rank of $c$

Until three cards of the same rank or know it is impossible
A diagnose-and-repair program

- three components 1, 2, 3;
- propositional symbol $ok_i$ : component $i$ is in working order;
- action $repair(i)$ : makes $ok_i$ true;
- action $test(i)$ : returns the truth value of $ok_i$;
- initial knowledge state: $K((ok_1 \leftrightarrow (ok_2 \land ok_3)) \land (\neg ok_1 \lor \neg ok_3))$;
- Goal: to have the three components working without replacing more components than necessary.

```
While $\neg K(ok_1 \land ok_2 \land ok_3)$ do
  $i :=$ smallest integer such that $\neg K ok_i$;
  If $\neg K \neg ok_i$ then $test(i)$ endif;
  If $K \neg ok_i$ then $replace(i)$ endif
Endwhile
```
Knowledge-based programs:

- introduced by Fagin, Halpern, Moses and Vardi [1995]
- studied for behaviour specification in distributed environments
- we use them as outputs of planning problems
- what are the benefits and pitfalls of using knowledge-based programs instead of standard programs?
Classical partially observable planning vs. Knowledge-based planning

Classical partially observable planning
Output = standard plan (policy):
- tree or DAG containing observations/actions
- branching on current state and observations

Knowledge-based planning
Output = knowledge-based program:
- branching conditions are subjective epistemic formulas
Example

- initial knowledge state: $O((\text{ok}_1 \leftrightarrow (\text{ok}_2 \land \text{ok}_3)) \land (\neg \text{ok}_1 \lor \neg \text{ok}_3))$
- goal knowledge state: $K(\text{ok}_1 \land \text{ok}_2 \land \text{ok}_3)$
- actions: test($i$), repair($i$) for $i = 1, 2, 3$

Knowledge-based plan:

\[\textbf{While } \neg K(\text{ok}_1 \land \text{ok}_2 \land \text{ok}_3) \textbf{ do} \]
\[\quad \text{find the smallest } i \text{ such that } \neg K \text{ok}_i; \]
\[\quad \textbf{If } \neg K \neg \text{ok}_i \textbf{ then test}(i); \]
\[\quad \textbf{If } K \neg \text{ok}_i \textbf{ then replace}(i) \]
\[\textbf{Endwhile} \]
Knowledge-based plans vs. policies

KBP

While $\neg K(ok_1 \land ok_2 \land ok_3)$ do
  find smallest $i$ such that $\neg Kok_i$;
  If $\neg K\neg ok_i$ then test($i$);
  If $K\neg ok_i$ then replace($i$)
Endwhile

standard policy

replace(1);
test(2);
If ok(2) then replace(3)
else replace(2);
test(3);
If $\neg ok(3)$ then replace(3)
endif
endif
Knowledge-based programs vs. standard programs

Knowledge-based programs:

- introduced by Fagin, Halpern, Moses and Vardi [1995]
- studied for behaviour specification in distributed environments
- we use them as *outputs of planning problems*
- *what are the benefits and pitfalls of using knowledge-based programs instead of standard programs?*
- [-] more difficult to execute than standard programs: evaluating branching conditions is computationally hard
- [+ ] more compact than standard programs
- [+ ] more natural to express than standard programs
Outline

Knowledge-based programs:

- introduced by Fagin, Halpern, Moses and Vardi [1995]
- studied for behaviour specification in distributed environments
- we use them as outputs of planning problems.

Our work:

- using knowledge-based programs as (single-agent) plans reaching some goals described by epistemic formulas
- LOFT-12 / ECAI-12: expressivity and complexity of plan verification
- TARK-13: comparing the succinctness of KBPs to that of standard plans + complexity of plan existence
- IJCAI-15: probabilistic knowledge-based programs
- ongoing work: KBP synthesis
- ongoing work: multi-agent KBP
Plan

Knowledge-based programs

Knowledge-based planning problems

Succinctness

KBP verification

KBP existence

Probabilistic KBPs

KBP synthesis

KBP synthesis

Multi-agent KBPs
Knowledge-based programs

Knowledge-based planning problems

Succinctness

KBP verification

KBP existence

Probabilistic KBPs

KBP synthesis

KBP synthesis

Multi-agent KBPs
Input

- set of propositional variables $X = \{x_1, \ldots, x_n\}$
  - Queen($c_1$), $ok_1$ ...
  - state = truth assignment (unobservable)

- set of actions

Knowledge-based program $\pi$:

- action, or
- sequence $\pi_1; \pi_2; \ldots; \pi_n$, or
- branching If $\Phi$ then $\pi_1$ else $\pi_2$, where $\Phi$ is a purely subjective S5 formula (Boolean combination of epistemic atoms $K\varphi$); or
- loop While $\Phi$ do $\pi_1$, where $\Phi$ is a purely subjective S5 formula.
Actions

Ontic action:
- changes the state of the world
- possibly nondeterministic + no feedback
- propositional symbol $x \mapsto \{x, x'\}$;
  - $x$ before the action is performed
  - $x'$ after the action is performed
- $\text{switch}(x_i): \Sigma = (x'_i \leftrightarrow \neg x_i) \land \land_{j \neq i}(x'_j \leftrightarrow x_j)$
- $x_i \leftarrow 0: \Sigma = (\neg x'_i)$
- $\text{reinit}(x_i): \Sigma = \land_{j \neq i}(x'_j \leftrightarrow x_j)$

Epistemic action:
- does not change the state of the world
- sends back one of several possible observations
- $\text{test}(x_i \lor x_j): \text{observe } x_i \lor x_j \text{ or observe } \neg(x_i \lor x_j)$
- $\text{ask-how-much-time-left}: \text{observe } (t = 15\text{mn}) \text{ or observe } (t = 10\text{mn})$
  - or observe $(t = 5\text{mn}) \text{ or observe } (t = 0)$
Executing a KBP

At every step :

- current state of variables $s^t$
  - $s^0 = x_1 x_2 \bar{x}_3$
- current knowledge state $M^t$
  - $M^t = \{ x_1 x_2 x_3, x_1 \bar{x}_2 x_3, x_1 x_2 \bar{x}_3 \}$
  - succinct representation $O(x_1 \land (x_2 \lor x_3)) : all I know is x_1 \land (x_2 \lor x_3)$.

Execution :

- branching condition / loop : evaluated in $M^t$
- ontic action : nondeterministic modification of $s^t$
- epistemic action :
  - no modification of $s^t$
  - reception of an observation $\omega$
Progression by an ontic action:

\[ M^t = \{ x_1x_2x_3, \bar{x}_1\bar{x}_2\bar{x}_3 \} \ O((x_1 \land x_2 \land x_3) \lor (\neg x_1 \land \neg x_2 \land \neg x_3)) \]

- progression of \( M^t \) by switch\((x_1)\):
  \[ M^{t+1} = \{ \bar{x}_1x_2x_3, x_1\bar{x}_2\bar{x}_3 \} \ O((\neg x_1 \land x_2 \land x_3) \lor (x_1 \land \neg x_2 \land \neg x_3)) \]

- progression of \( M^{t+1} \) by reinit\((x_1)\):
  \[ M^{t+2} = \{ x_1x_2x_3, \bar{x}_1x_2x_3, x_1\bar{x}_2\bar{x}_3, \bar{x}_1\bar{x}_2\bar{x}_3 \} \ O(x_2 \leftrightarrow x_3) \]

Progression by an observation (received after some epistemic action):

- action test\((x_1 \land x_2)\), observation \(\neg(x_1 \land x_2)\):

- progression of \( M^{t+2} \) by observation \(\neg(x_1 \land x_2)\):
  \[ M^{t+3} = \{ \bar{x}_1x_2x_3, x_1\bar{x}_2\bar{x}_3, \bar{x}_1\bar{x}_2\bar{x}_3 \} \ O((x_2 \leftrightarrow x_3) \land \neg(x_1 \land x_2)) \]
Knowledge-based programs

Knowledge-based planning problems

Succinctness

KBP verification

KBP existence

Probabilistic KBPs

KBP synthesis

KBP synthesis

Multi-agent KBPs
Classical planning

▶ Set of initial states and goal states (described succinctly)
▶ Set of actions whose effects are described succinctly
▶ Output: standard plan (policy):
  ▶ tree or DAG containing observations/actions
  ▶ branching on current state and observations
Knowledge-based planning problems

- **Initial knowledge state** $M^0$:
  - possibly $\top$
  - must contain the true initial state

- **Goal** $G$ (purely subjective epistemic formula)

- **Valid plan** if
  - terminates
  - for every possible sequence of states $s^0 \in M^0 \ldots s^{\text{final}} \in M^{\text{final}}$
  we have $s^{\text{final}} \models G$
Example

- initial knowledge state: $O((ok_1 \leftrightarrow (ok_2 \land ok_3)) \land (\neg ok_1 \lor \neg ok_3))$
- goal knowledge state: $K(ok_1 \land ok_2 \land ok_3)$
- actions: test($i$), repair($i$) for $i = 1, 2, 3$

Knowledge-based plan:

```plaintext
While $\neg K(ok_1 \land ok_2 \land ok_3)$ do
    find the smallest $i$ such that $\neg K ok_i$
    If $\neg K \neg ok_i$ then test($i$)
    If $K \neg ok_i$ then replace($i$)
Endwhile
```
Knowledge-based plans vs. standard policies

- A standard policy is a KBP in which the last action executed before any branching condition \( \text{if } \Phi \) or \( \text{while } \Phi \) is an epistemic action \( a \) such that \( \Phi \) is one of the possible observations for \( a \).

- For every KBP \( \pi \) there exists a standard policy \( \pi' \) “equivalent” to \( \pi \) (\( \pi \) and \( \pi' \) have the same execution traces).

Expressivity:

- there exists a valid knowledge-based for a planning problem \( P \) iff there exists a valid standard policy for \( P \)
Knowledge-based plans vs. policies

KBP

\[
\begin{align*}
\text{While } & \neg K(ok_1 \land ok_2 \land ok_3) \text{ do} \\
& \text{find smallest } i \text{ such that } \neg K ok_i ; \\
& \text{If } \neg K \neg ok_i \text{ then } test(i) ; \\
& \text{If } K \neg ok_i \text{ then } replace(i) \\
\text{Endwhile}
\end{align*}
\]

standard policy

\[
\begin{align*}
\text{replace}(1) ; \\
\text{test}(2) ; \\
\text{If } & ok(2) \\
\text{then } & replace(3) \\
\text{else } & replace(2) ; \\
& test(3) ; \\
& \text{If } \neg ok(3) \\
& \text{then } replace(3) \\
\text{endif}
\end{align*}
\]
On-line execution:

- **standard policy**:
  - move to the subtree corresponding to the observation and execute the next action
  - constant time

- **knowledge-based plan**:
  - branching / loop condition: decide $M^t \models \Phi$
  - NP-hard and coNP-hard, in $\Delta_2^P$
Proposition: unless \( \text{NP} \subseteq \text{P}/\text{poly} \) (extremely unlikely), while-free KBPs with atomic branching conditions are exponentially more succinct than while-free standard policies.

Proof sketch:

- For each \( n \in \mathbb{N} \) we build a polysize KBP \( \pi_n \) that “reads” a CNF formula \( \varphi \) and either makes sure that it is unsatisfiable or else builds a model of it.

- If there is a family of standard policies \( \pi'_n \) for every \( n \), of size polynomial in \( |\pi_n| \), with \( \pi_n \) equivalent to \( \pi'_n \), then there is a (possibly nonuniform) polytime algorithm for 3SAT, yielding \( \text{NP} \subseteq \text{P}/\text{poly} \).
Proposition: KBPs (with loops) are more succinct than standard policies (with loops).

Proof sketch:

- there is a polynomial \( pol \) and a collection of KBPs \( (\pi_n)_n \) such that \( |\pi_n| \leq pol(n) \) and such that \( \pi_n \) “counts” up to \( 2^{2^n} - 1 \) (by going once through all knowledge states).

- we build a family of planning problems \( (P_n)_n \) such that the only valid plans for \( P_n \) are all equivalent to \( \pi_n \).

- assume that for all \( n \) there is a standard policy \( \pi'_n \) for \( P_n \) and \( |\pi'_n| \leq pol(n) \) ; then \( \pi'_n \) can manipulate only \( pol(n) \) variables, and can have only \( 2^{pol(n)}.|\pi'_n| \) configurations (states + control points) ; then it cannot count up to \( 2^{2^n} - 1 \), contradiction.
Knowledge-based plans vs. policies: succinctness

Proposition: KBPs are more succinct than while-free KBPs.

Proof sketch: later
Knowledge-based programs

Knowledge-based planning problems

Succinctness

KBP verification

KBP existence

Probabilistic KBPs

KBP synthesis

KBP synthesis

Multi-agent KBPs
KBP existence vs. KBP verification

KBP verification

input \( P = (\text{initial belief state, actions, goal}) \)

question is \( \pi \) valid for \( P \) ?

KBP existence

input \( P = (\text{initial belief state, actions, goal}) \)

question is there a valid KBP \( \pi \) for \( P \) ?

small KBP existence

input \( P = + \text{ integer } k \text{ encoded in unary} \)

question is there a valid KBP \( \pi \) for \( P \) such that \( |\pi| \leq k \) ?
KBP verification: overview of results

loop-free programs

- $\Pi_2^P$-complete ($\Pi_2^P = \text{coNP}^{\text{NP}}$)
- remains $\Pi_2^P$-complete with each of the following restrictions:
  - ontic actions only
  - epistemic actions only
- standard plan verification: coNP-complete

programs with loops

- EXPSPACE-complete
- remains EXPSPACE-complete even if we know that $\pi$ terminates
- standard plan verification: PSPACE-complete
KBP verification: loop-free programs

- $\Pi^P_2$-complete ($\Pi^P_2 = \text{coNP}^{\text{NP}}$)
- Hardness proof easy
- Membership proof based on the following nondeterministic algorithm that shows that a plan is \textit{not} valid:
  - guess a sequence of observations
  - at each step with a branching condition $\Phi$, evaluate $\Phi$ [Needs a polynomial number of NP oracles]
  - check that the goal is not satisfied at the end of the execution
KBP verification: general programs

- EXPSPACE-complete
- key point: a loop can be executed up to $2^{2^n} - 2$ times (visit all possible belief states)
- membership easy
- hardness by reduction from NONDETERMINISTIC UNOBSERVABLE PLAN EXISTENCE (Haslum and Jonsson, 99)
Proposition: KBPs are more succinct than while-free KBPs.

Proof sketch:

- verifying a KBP with loops is EXPSPACE-complete;
- verifying a while-free KBP is $\Pi_2^P$-complete;
- $\Pi_2^P \subseteq \text{PSPACE} \subset \text{EXPSPACE}$ (strict inclusion, Savitch’s theorem)
## KBB existence: overview of results

<table>
<thead>
<tr>
<th></th>
<th>unbounded</th>
<th>bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>2-EXPTIME-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>while-free</td>
<td>2-EXPTIME-complete</td>
<td>$\Sigma_3^p$-complete</td>
</tr>
<tr>
<td>ontic</td>
<td>EXPSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>while-free, ontic</td>
<td>EXPSPACE-complete</td>
<td>$\Sigma_2^p$-complete</td>
</tr>
<tr>
<td>while-free, epistemic</td>
<td>PSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>while-free, epistemic, positive goal</td>
<td>coNP-complete</td>
<td>$\Sigma_2^p$-complete</td>
</tr>
</tbody>
</table>
### KBP existence

<table>
<thead>
<tr>
<th></th>
<th>bounded</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>2-EXPTIME-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>while-free</td>
<td>2-EXPTIME-complete</td>
<td>$\Sigma_3^P$-complete</td>
</tr>
<tr>
<td>ontic</td>
<td>EXPSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>while-free, ontic</td>
<td>EXPSPACE-complete</td>
<td>$\Sigma_2^P$-complete</td>
</tr>
<tr>
<td>while-free, epistemic</td>
<td>PSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>while-free, epistemic,</td>
<td>coNP-complete</td>
<td>$\Sigma_2^P$-complete</td>
</tr>
<tr>
<td>positive goal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Corollaries from known results in planning together with the fact that there exists a KBP for a planning problem off there exists a standard plan.
KBP existence

<table>
<thead>
<tr>
<th></th>
<th>unbounded</th>
<th>bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>2-EXPTIME-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>while-free</td>
<td>2-EXPTIME-complete</td>
<td>$\Sigma^p_3$-complete</td>
</tr>
<tr>
<td>ontic</td>
<td>EXPSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>while-free, ontic</td>
<td>EXPSPACE-complete</td>
<td>$\Sigma^p_2$-complete</td>
</tr>
<tr>
<td>while-free, epistemic</td>
<td>PSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>while-free, epistemic, positive goal</td>
<td>coNP-complete</td>
<td>$\Sigma^p_2$-complete</td>
</tr>
</tbody>
</table>

- **membership**: guess $\pi$ of size $\leq k$ and verify it; \textsc{Plan Verification} is in EXPSPACE and $\text{NEXPSPACE} = \text{EXPSPACE}$.

- **hardness**: reduction from \textsc{Plan Verification}. Build a planning problem $P'$, and let $k = |\pi|$, such that every valid plan for $P'$ is equivalent to $\pi$. 
### KBP existence

<table>
<thead>
<tr>
<th></th>
<th>unbounded</th>
<th>bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>(2)EXPTIME-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>while-free</td>
<td>(2)EXPTIME-complete</td>
<td>(\Sigma^p_3)-complete</td>
</tr>
<tr>
<td>ontic</td>
<td>EXPSPACE-complete</td>
<td>(\Sigma^p_3)-complete</td>
</tr>
<tr>
<td>while-free, ontic</td>
<td>EXPSPACE-complete</td>
<td>(\Sigma^p_2)-complete</td>
</tr>
<tr>
<td>while-free, epistemic</td>
<td>PSPACE-complete</td>
<td>(\Sigma^p_2)-complete</td>
</tr>
<tr>
<td>while-free, epistemic, positive goal</td>
<td>coNP-complete</td>
<td>(\Sigma^p_2)-complete</td>
</tr>
</tbody>
</table>

- **membership**: guess \(\pi\) and verify it; \(\text{PLAN VERIFICATION}\) is in \(\Pi^p_2\).
- **hardness**: reduction from \(\text{QBF}_{3,\exists}\).
## KBP existence

<table>
<thead>
<tr>
<th></th>
<th>Unbounded</th>
<th>Bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>2-EXPTIME-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>While-free</td>
<td>2-EXPTIME-complete</td>
<td>(\Sigma^p_3)-complete</td>
</tr>
<tr>
<td>Ontic</td>
<td>EXPSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>While-free, ontic</td>
<td>EXPSPACE-complete</td>
<td>(\Sigma^p_2)-complete</td>
</tr>
<tr>
<td>While-free, epistemic</td>
<td>PSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>While-free, epistemic, positive goal</td>
<td>coNP-complete</td>
<td>(\Sigma^p_2)-complete</td>
</tr>
</tbody>
</table>

Branching is not necessary because we never get any feedback; the problem is equivalent to polynomially-bounded plan existence without branching, which is \(\Sigma^p_2\)-complete.
### KBP existence

<table>
<thead>
<tr>
<th></th>
<th>unbounded</th>
<th>bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>2-EXPTIME-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>while-free</td>
<td>2-EXPTIME-complete</td>
<td>$\Sigma^p_3$-complete</td>
</tr>
<tr>
<td>ontic</td>
<td>EXPSPACE-complete</td>
<td>$\Sigma^p_3$-complete</td>
</tr>
<tr>
<td>while-free, ontic</td>
<td>EXPSPACE-complete</td>
<td>$\Sigma^p_2$-complete</td>
</tr>
<tr>
<td>while-free, epistemic</td>
<td>PSPACE-complete</td>
<td>$\Sigma^p_2$-complete</td>
</tr>
<tr>
<td>while-free, epistemic, positive goal</td>
<td>coNP-complete</td>
<td>$\Sigma^p_2$-complete</td>
</tr>
</tbody>
</table>

- **membership**: because an epistemic action needs to be executed at most once, if a planning problem has a valid KBP then it has a valid KBP of height bounded by the number of epistemic actions.
- **+ searching a polynomial-height tree can be done in PSPACE.**

- **hardness**: reduction from QBF.
### KBP existence

<table>
<thead>
<tr>
<th></th>
<th>unbounded</th>
<th>bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>2-EXPTIME-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>while-free</td>
<td>2-EXPTIME-complete</td>
<td>( \Sigma^p_3 )-complete</td>
</tr>
<tr>
<td>ontic</td>
<td>EXPSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>while-free, ontic</td>
<td>EXPSPACE-complete</td>
<td>( \Sigma^p_2 )-complete</td>
</tr>
<tr>
<td>while-free, epistemic</td>
<td>PSPACE-complete</td>
<td>?</td>
</tr>
<tr>
<td>while-free, epistemic, positive goal</td>
<td>coNP-complete</td>
<td>( \Sigma^p_2 )-complete</td>
</tr>
</tbody>
</table>

- membership, unbounded: performing an epistemic action cannot harm; there exists a valid KBP iff the KBP consisting in performing all epistemic actions in any order is valid.
- membership, bounded: guess a set of \( k \) epistemic actions and perform them; verification is in coNP.
- hardness: reductions from UNSAT and QBF\(_2,\exists\).
5 doors:
- a tiger hidden behind two of them
- a princess behind one of the other three
- initially, all possible configurations equiprobable.

Sensing actions \(\text{listen}_i, \ i = 1, \ldots, 4\) (not 5). Feedback:
- if a tiger is behind door \(i\): hear the tiger roaring \((r_+)\) with probability 0.5, or not \((r_-)\) with probability 0.5
- if no tiger behind door \(i\): \(r_-\) with probability 1;

Ontic actions \(\text{open}_i\) : \(i = 1, \ldots, 5\). Effects: the agent...
- ... becomes eaten by the tiger if there is one behind door \(i\) (reward \(-1\))
- ... becomes married to the princess if she is behind door \(i\) (reward \(+1\))
π:

\[
\begin{align*}
\text{listen}_1; \text{listen}_2; \text{listen}_3; \text{listen}_4; \\
\text{while } P(t_1) > 0.1 \land \cdots \land P(t_5) > 0.1 \text{ do} \\
\text{if } P(t_1) \leq P(t_2) \land \cdots \land P(t_1) \leq P(t_5) \text{ then} \\
\quad [ \text{listen}_1; \text{if } P(t_1) \leq 0.1 \text{ then open}_1 ] \\
\text{elseif } P(t_2) \leq P(t_1) \land \cdots \land P(t_2) \leq P(t_5) \text{ then} \\
\quad [ \text{listen}_2; \text{if } P(t_2) \leq 0.1 \text{ then open}_2 ] \\
\text{\ldots} \\
\text{else [ if } P(t_5) \leq 0.1 \text{ then open}_5 ]
\end{align*}
\]

π corresponds to a (less succinct) POMDP policy, with branching on sequences of observations.
Informally: maintain a list $L$ of pairs $⟨K\varphi, a⟩$ such that performing $a$ in knowledge state $K\varphi$ eventually leads to the goal

1. $L$ initialized to $\{⟨K\varphi, \text{stop}⟩ | K\varphi \in \Gamma\}$
2. repeat
   1. $\Gamma’ = \bigvee \{K\varphi | ⟨K\varphi, a⟩ \in L \text{ for some } a\}$
   2. regress $\Gamma’$ by some action $\alpha$
   3. add $⟨\text{Reg}(\Gamma’, \alpha), \alpha⟩$ to $L$ (unless it is redundant)
3. until the initial knowledge state implies $K\varphi$ for some $⟨K\varphi, a⟩$ in $L$

If $L = \{⟨\varphi_i, \alpha_i⟩, i = 1, \ldots, m\}$, return

REPEAT
  CASE
    $\varphi_1 : \alpha_1$
    ...
    $\varphi_m : \alpha_m$
  END
UNTIL stop
KBP synthesis

Same example as in (Herzig, Lang & Marquis, 2003):

- two propositional variables \( u, v \)
- epistemic actions \( \alpha = \text{test}(u \land v), \beta = \text{test}(u \leftrightarrow v) \)
- ontic action \( \gamma = \text{switch}(u) \)
- initial knowledge state \( K\top \)
- goal \( Kv \lor K\neg v \).

Successive values of \( L \):

1. initially: \( L = \{\langle Kv, \text{stop} \rangle, \langle K\neg v, \text{stop} \rangle\} \)
2. add \( \langle K(v \rightarrow u), \alpha \rangle \)
3. add \( \langle K(v \rightarrow \neg u), \gamma \rangle \)
4. add \( \langle K\top, \beta \rangle \)

\[
L = \{\langle Kv, \text{stop} \rangle, \langle K\neg v, \text{stop} \rangle, \langle K(v \rightarrow u), \alpha \rangle, \langle K(v \rightarrow \neg u), \gamma \rangle, \langle \top, \alpha \rangle \}
\]
The plan returned is

```
REPEAT
  CASE
    \(K_v\) : \(\text{stop}\)
    \(K \neg v\) : \(\text{stop}\)
    \(K(v \rightarrow u)\) : \(\alpha\)
    \(K(v \rightarrow \neg u)\) : \(\gamma\)
    \(K\top\) : \(\beta\)
  END
UNTIL \(\text{stop}\)
```
Multi-agent KBPs : three prisoners and a lightbulb

The propositional variables:

- $\text{in}(i) : i$ is in the room
- $\text{hasbeen}(i) : i$ has already been in the room
- $\text{light} :$ the light is switched on
- $\text{success}$
- $\text{end}$ (ensures $\text{tell}$ is performed successfully at most once)
Multi-agent KBPs: three prisoners and a lightbulb

The actions:

- **wait**($i$): nature possibly sends one of the agents into the room; $i$ learns whether he is in the room or not; $i$ can be sent in the room if it is not empty; $i$ forgets about the light if he knew something about it.

$$ (K_i \text{in}(i) \lor K_i \neg \text{in}(i)) \land K_i \bigwedge_{j\neq k} (\text{in}(j) \rightarrow (\neg \text{in}(k) \land \neg \text{in}(k)')) \land (\ldots) $$

- **observe**($i$): $i$ learns the value of $l$, provided that he is in the room:

$$ K_i(\text{in}(i) \rightarrow \text{light}) \lor K(\text{in}(i) \rightarrow \neg \text{light}) \land (\ldots) $$

- **switch**: $K_i(l' \leftrightarrow \neg l) \land (\ldots)$

- **exit**($i$): $i$ exits the room if he was in it: $K_i \neg \text{in}(i) \land (\ldots)$

- **tell**: $K_i (\text{end}' \land (\text{hasbeen}(1) \land \text{hasbeen}(2) \land \neg \text{end} \rightarrow \text{success}') \land (\ldots))$

and all these actions theories are common knowledge
Multi-agent KBPs: three prisoners and a lightbulb

1. $\pi_0:$

2:

3. if $K_0 \neg in(0)$ then
4. wait(0)
5. else
6. observe(0);
7. if $K_0(hasbeen(1) \land hasbeen(2))$ then
8. tell
9. else
10. if $K_0 light$ then
11. switch
12. end if
13. end if
14. exit
15. end if
Multi-agent KBPs: three prisoners and a lightbulb

1: \( \pi_1 \) :
2:
3: \textbf{if} \( K_1 \neg in(1) \) \textbf{then}
4: \hspace{1em} wait(1)
5: \textbf{else}
6: \hspace{1em} \textbf{if} \( K_1 \neg hasbeen(1) \) \textbf{then}
7: \hspace{2em} observe(1);
8: \hspace{2em} \textbf{if} \( K_1 \neg light \) \textbf{then}
9: \hspace{3em} switch
10: \hspace{3em} \textit{hasbeen}(1) := true
11: \hspace{2em} \textbf{end if}
12: \hspace{1em} \textbf{end if}
13: \hspace{1em} \textbf{end if}
14: \hspace{1em} \textbf{end if}

\( \pi_2 \) is the same as \( \pi_2 \), replacing 1 by 2 everywhere.