PKS (Planning with Knowledge and Sensing) is a “knowledge-level” planner that is able to construct conditional plans in the presence of incomplete knowledge and sensing (Bacchus and Petrick 1998; Petrick and Bacchus 2002; 2004). The key idea of this approach is to represent the agent’s knowledge state with a first-order language, and to represent actions by their effects on the agent’s knowledge, rather than by their effects on the environment. Since general reasoning in such a rich language is impractical, PKS employs a restricted subset of the language and a limited amount of inference in that subset. As a result, PKS includes non-propositional features, such as functions and variables. The knowledge-based approach contrasts some of the alternate trends that have concentrated on propositional representations over which complete reasoning is feasible. Such works often represent the set of all possible worlds (i.e., the set of all states compatible with the agent’s incomplete knowledge) using various techniques (e.g., BDDs, Graphplan-like structures, or clausal representations). These techniques yield planning systems that are able to generate plans requiring complex combinatorial reasoning.

By representing problems at the knowledge level, PKS can generate plans that are often quite “natural” and have a simple structure. Furthermore, PKS can often “abstract” away from some of the irrelevant distinctions that occur at the world level. Compared to the possible-worlds approaches, this higher-level representation is richer, but the inferences it supports are weaker. Nevertheless, PKS is able to solve problems that cannot be solved by alternate approaches.

We briefly describe some of the important components we have implemented in the PKS system.

**Knowledge representation:** PKS’s knowledge (rather than the state of the world) is represented by a set of four databases. Any configuration of the databases corresponds to a collection of modal logic formulae that precisely characterizes PKS’s knowledge state. To ensure an efficient inference mechanism, we restrict the types of knowledge (especially disjunctive knowledge) that can be modelled:

*\( K_f \):* The first database stores both positive and negative facts, but does not employ a closed world assumption. *\( K_f \) can include any ground literal, \( \ell \); \( \ell \in K_f \) means that \( \ell \) is known. *\( K_f \) can also contain knowledge of function values.

*\( K_w \):* The second database models the plan-time effects of sensing actions. \( \phi \in K_w \) means that at plan time the planner either knows \( \phi \) or knows \( \neg \phi \), and that at execution time this disjunction will be resolved. PKS uses such “know-whether” facts to construct conditional branches in a plan.

*\( K_v \):* The third database stores information about function values that will become known to PKS at execution time. *\( K_v \) can contain any unnested function term; such terms model the plan-time effects of sensing actions that return numeric values. PKS can use *\( K_v \) knowledge of finite-range functions to insert multi-way branches into a plan.*

*\( K_e \):* The fourth database contains “exclusive-or” knowledge of literals. Entries in *\( K_e \) have the form \( \ell_1|\ell_2|\ldots|\ell_n \), where each \( \ell_i \) is a ground literal. Such a formula represents knowledge of the fact that “exactly one of the \( \ell_i \) is true.” Such knowledge is common in many planning scenarios.

**Actions:** Actions in PKS are modelled as updates to the databases (i.e., knowledge state), rather than as updates to the world state. Actions may be parameterized and can have conditional effects. An action, but incomplete, inference algorithm determines if an action’s preconditions hold, by examining the database contents to draw conclusions about what PKS does and does not know or “know whether.” Applying an action’s effects simply involves adding or deleting the appropriate formulae from the collection of databases.

Table 1 shows two PKS actions. *pour-on-lawn* pours a liquid onto a lawn, with the effect that if the liquid is poisonous, the lawn becomes dead. *pour-on-lawn* has two conditional effects: if *poisonous* is not known, delete *lawn-dead* from *\( K_f \); if *poisonous* is known, then add *lawn-dead* to *\( K_f \).* *sense-lawn* senses whether or not the lawn is dead. It is represented as an update that adds *lawn-dead* to *\( K_w \).*

**Conditional plans:** A PKS conditional plan is a tree whose nodes are labelled by a knowledge state, and whose edges are labelled by an action or a sensed fluent. An existing plan is extended in a forward-chaining manner by adding a new...
Temporal extended goals: Inference algorithm at a given knowledge state. A primitive from a set of primitive queries that can be evaluated by the standard inference algorithm, by examining the effects—and non-effects—of actions in a conditional plan. Linearizations differ in how the plan proceeds: each path to a leaf becomes a linear sequence of states and actions (the plan). Linearizations differ in how the plan proceeds: each path to a leaf becomes a linear sequence of states and actions (the plan). Linearizations differ in how the plan proceeds: each path to a leaf becomes a linear sequence of states and actions (the plan).

Postdiction: Although PKS is able to efficiently generate plans, there are situations where the resulting knowledge states fail to contain some “intuitive” conclusions. As a result, PKS also uses a postdiction (Sandewall 1994) procedure to extend the standard inference algorithm, by examining the effects—and non-effects—of actions in a conditional plan. PKS considers linearizations of a conditional plan: each path to a leaf becomes a linear sequence of states and actions (the plan). Linearizations differ in how the knowledge resolves itself during execution, and how this resolution affects the actions PKS subsequently executes. For each linearization, we apply a simple set of (sound) backward and forward inferences to draw additional conclusions along that sequence. These inferences potentially augment any of the knowledge states in the linearization. A goal is satisfied if it is satisfied in every linearization of the plan.

The postdiction inferences are often essential for PKS to successfully generate plans. E.g., Figure 1 illustrates a conditional plan, and its two linearizations, generated by PKS using the actions from Table 1. The additional inferences produced by postdiction are shown in bold. Using postdiction, PKS can prove that in every outcome of the plan it either knows poisonous or knows ¬poisonous. Without postdiction the conditional plan does not satisfy these conclusions.

Numerical evaluation: PKS includes extensive support for numeric expressions and can construct plans to manage limited resources or satisfy certain numeric constraints. Currently, PKS can only deal with numeric expressions containing terms that can be evaluated down to a number at plan time. Even with this restriction, numeric expressions can be quite complex: PKS permits a subset of the set of C language expressions. Specifically, numeric expressions can contain all of the standard arithmetic operations, logical connective operators, and limited control structures (e.g., conditional evaluations and simple iterative loops). Temporary variables may also be introduced into calculations. Numeric expressions can be used in database updates, queries, and goals.

Planning problems: We have tested PKS on a variety of planning problems (see Petrick and Bacchus 2002; 2004) for details and empirical results. Many of the “classical” conditional planning problems (e.g., bomb in the toilet, medicate) become trivial when modelled at the knowledge level. Other domains (e.g., opening a safe, painted door) illustrate the advantage of being able to manipulate functions, and result in “natural” plans. We have also experimented with a series of problems taken from the UNIX domain. In particular, these domains have motivated the development of some of PKS’s components (e.g., temporally extended goals, numerical expressions). Moreover, these examples illustrate the utility of the knowledge-based approach to planning with incomplete knowledge. We believe that this approach continues to have great potential for building powerful planners.

References


