Third International Symposium on Foundations of Information and Knowledge Systems (FoIKS) , 2004

Plan Databases: Model and Algebra

Fusun Yaman[1], Sibel Adali[2], Dana Nau[1], Maria L. Sapino[3], and V.S. Subrahmanian[1]

[1] University of Maryland, College Park MD,20705, USA
{fusun,nau,vs}@cs.umd.edu
[2] Rensselaer Polytechnic Inst.,Troy, NY, 12180, USA
sibel@cs.rpi.edu
[3] Università di Torino, C.So Svizzera, 185-10149,Torino, Italy
mlsapino@di.unito.it

Abstract. Despite the fact that thousands of applications manipulate plans, there has been no work to date on managing large databases of plans. In this paper, we first propose a formal model of plan databases. We describe important notions of consistency and coherence for such databases. We then propose a set of operators similar to the relational algebra to query such databases of plans.

1 Introduction

Most of AI planning has focused on creating plans. However, the complementary problem of querying a collection of plans has not been studied. Querying plans is of fundamental importance in today's world. Shipping companies like UPS and DHL create plans for each package they have to transport. Multiple programs and humans need to query the resulting database of plans in order to determine how to allocate packages to specific drivers, to identify choke areas in the distribution network, and to determine which facilities to upgrade, etc. Likewise, every commercial port creates detailed plans to route ships into the port. Port officials need to determine which ships are on schedule, which ships are off schedule, which ships may collide with one another (given that one of them is off schedule) and so on. A similar application arises in the context of air traffic control - prior to takeoff, every flight has a designated flight plan and flight path. It is not uncommon for planes to be off schedule and/or off their assigned path. In other words, the plane may not be at the assigned location at the assigned time. Maintaining the integrity of air traffic corridors, especially in heavily congested areas (e.g. near Frankfurt or London airport), is a major challenge. Air traffic controllers need to be able to determine which flights are on a collision course, which flights are not maintaining adequate separation, which flights may intrude onto another flight's airspace and when.

These are just three simple applications where we need the ability to query collections of plans. *We emphasize that in this paper, we are not interested in creating plans, just in querying them.* The long version of this paper discusses

issues such as how to update databases of plans (which does involve some planning).

In this paper, we develop a formal model of a **plan database.** We then describe two important properties of such databases - **consistency and coherence** and present results that these properties are polynomially checkable. We then present a relational-algebra style **plan algebra** to query plan databases. In addition to the relational style operators, our algebra contains operators unique to plan databases.[4]

2 Plan Database Model

In this section, we introduce the basic model of a plan database. The concept of a plan is an adaptation of the notion of a plan in the well known PDDL planning language [4].

Definition 1. *A **planspace**, $\mathcal{PS}$, is a finite set of relations. A **planworld** pw over a planspace $\mathcal{PS}$ is a finite instance of each relation in $\mathcal{PS}$.*

We use the standard notion of a relational schema and domain of an attribute when describing planspaces. There are certain special relations called *numeric relations*.

Definition 2. *A **numeric relation** in a planspace is a relation $R(A_1, \ldots, A_n, V)$ where $\langle A_1, \ldots, A_n \rangle$ forms a primary key and V is of type real or integer.*

Note that a numeric relation $R(A_1, \ldots, A_n, V)$ represents a function f_R that maps $dom(A_1) \times \ldots \times dom(A_n) \rightarrow dom(V)$.

Example 1 (Package Example). A shipping company's planspace may use the following relations to describe truck locations, truck drivers, packages and valid routes:

- *at(object, location)*: specifies the location of drivers, trucks and package.,
- *route(location1, location2)*: specifies that there is a viable route from *location1* to *location2*.
- *in(package, truck)*: specifies information about which truck carries which package.
- *driving(driver, truck)*: specifies who is driving which truck.
- *fuel(truck, level)*: a numerical relation specifying the fuel level of each truck.

2.1 Actions

In this section, we define two types of actions, simple actions that occur instantaneously, and durative actions that take place over a period of time and

[4] Due to space constraints, we are unable to present operators to update plan databases.

possibly require certain conditions to be true during this time. A *term* over A_i is either a member of $dom(A_i)$ or a variable over $dom(A_i)$. If $R(A_1, \ldots, A_n)$ is a relation in a planspace and t_i $(1 \leq i \leq n)$ is a term over A_i, then $R(t_1, \ldots, t_n)$ is an atom. Likewise, if A_i, A_j are attributes, then $A_i = A_j$ is an atom - if A_i, A_j are both attributes such that $dom(A_i), dom(A_j)$ are subsets of the reals, then $A_i \leq A_j, A_i < A_j, A_i \geq A_j$ and $A_i > A_j$ are atoms. If A is an atom, then A and $\neg A$ are literals.

If $R(A_1, \ldots, A_n, V)$ is a numeerical relation and δ is a real number, then $incr(R, \delta), decr(R, \delta)$ and $assign(R, \delta)$ are *nu-formulas*. These formulas say that the V column of R must be increased or decreased by (or set to) the value δ.

Definition 3. *A* ***simple action*** *w.r.t. a planspace $\mathcal{PS}$ is a 5-tuple consisting of*

- *Name: A string $\alpha(A_1, \ldots, A_n)$, where each A_i is a variable called a* parameter *of the action.*
- Precondition $pre(\alpha)$*: a conjunction of literals over the planspace.*
- Add list $add(\alpha)$*: set of atoms (denote what becomes true after executing the action).*
- Delete list $del(\alpha)$*: set of atoms (denote what becomes false after executing the action).*
- Numeric update list $update(\alpha)$*: set of nu-formulas.*

If $c_i \in dom(A_i)$ for $1 \leq i \leq n$, then $\alpha(c_1, \ldots, c_n)$ is an ***instance*** *of $\alpha(A_1, \ldots, A_n)$. As is standard practice in AI planning, we assume that all actions are* range restricted*, i.e., that all variables appearing in the condition and effects are parameters of the action. Thus, each action is unambiguously specified by its name.*

A simple action is executed instantaneously (in 0 time) and its effects are described by its add list, delete list and update list. We assume the nu-formulas in the numeric update list are executed in the following order: all *incr* updates first, then all *decr* updates, and then all *assign* updates.

Definition 4. *Suppose $\alpha(\boldsymbol{t})$ is an action instance. $\mathsf{L}(\alpha(\boldsymbol{t}))$ denotes the set of all numerical variables updated by $\alpha(\boldsymbol{t})$, $\mathsf{R}(\alpha(\boldsymbol{t}))$ denotes the set of numerical variables read by $\alpha(\boldsymbol{t})$ and $\mathsf{L}^*(\alpha(\boldsymbol{t}))$ is the set of all numerical variables whose value is being increased or decreased (but not assigned) by $\alpha(\boldsymbol{t})$.*

A simple action instance $\alpha(\boldsymbol{t})$ is *executable* in planworld pw if $pre(\alpha(\boldsymbol{t}))$ is satisfied by pw. The concept of mutual exclusion of actions specifies when two actions cannot co-occur.

Definition 5. *Two simple action instances $\alpha(\boldsymbol{t})$ and $\beta(\boldsymbol{z})$ are* ***mutually exclusive*** *if any of the following hold:*

$pre(\alpha(\boldsymbol{t})) \cap (add(\beta(\boldsymbol{z})) \cup del(\beta(\boldsymbol{z}))) \neq \emptyset$ $\quad$ $pre(\beta(\boldsymbol{z})) \cap (add(\alpha(\boldsymbol{t})) \cup del(\alpha(\boldsymbol{t}))) \neq \emptyset$

$add(\alpha(\boldsymbol{t})) \cap del(\beta(\boldsymbol{z})) \neq \emptyset$ $\quad$ $del(\alpha(\boldsymbol{t})) \cap add(\beta(\boldsymbol{z})) \neq \emptyset$

$\mathsf{L}(\alpha(\boldsymbol{t})) \cap \mathsf{R}(\beta(\boldsymbol{z})) \neq \emptyset$ $\quad$ $\mathsf{L}(\beta(\boldsymbol{z})) \cap \mathsf{R}(\alpha(\boldsymbol{t})) \neq \emptyset$

$\mathsf{L}(\alpha(\boldsymbol{t})) \cap \mathsf{L}(\beta(\boldsymbol{z})) \neq \mathsf{L}^*(\alpha(\boldsymbol{t})) \cap \mathsf{L}^*(\beta(\boldsymbol{z}))$

$\alpha(\boldsymbol{t})$ *and* $\beta(\boldsymbol{z})$ *are* mutually compatible *(i.e. they can occur at the same time) if they are not mutually exclusive.*

If A is an atom (resp. nu-formula, literal, conjunction of literals) and $w \in \{AtStart, AtEnd, OverAll\}$ then $w : A$ is an *annotated atom* (resp. nu-formula, literal, conjunction of literals).

Definition 6. *A* ***durative action*** *w.r.t. a planspace $\mathcal{PS}$ is a 5-tuple consisting of*

- Name: *this is the same as for simple actions.*
- Condition *cond*(α)*: set of annotated literals.*
- Add List *add*(α)*: set of $w : A$ where A is an atom and $w \in \{AtStart, AtEnd\}$.*
- Delete list *del*(α)*: set of $w : A$ where A is an atom and $w \in \{AtStart, AtEnd\}$.*
- Numeric update list *update*(α)*: set of $w : A$ where A is a nu-formula and $w \in \{AtStart, AtEnd\}$.*

Instances of durative actions are defined in the same way as for simple actions.

An ***action instance*** is an expression of the form $\alpha(\boldsymbol{t})$ for a vector $\boldsymbol{t}$ of the form $(c_1, \ldots, c_n)$ that assigns a constant to each variable in the name.

Example 2. We present a durative action, *load-truck(p,t,l)*, which loads package p into truck t at location l.

- *cond*(*load-truck(p,t, l)*) = {*atStart* : *at(p,l)*, *atStart* : *at(t,l)*, *overAll* : *at(t,l)*}. This says that when we start executing the *load-truck(p,t,l)* action, the truck and package should both be at location l and that throughout the execution of the *load-truck* action, the truck must be at location l (e.g. it cannot start moving during the loading operation).
- *add*(*load-truck(p,t,l)*) = {*atEnd* : *in-truck(p,t)*}. This says that the atom *in-truck*(p, t) is true at the end of the loading operation.
- *del*(*load-truck(p,t,l)*) = {*atStart* : *at(p,l)*}. Once we start executing the action, package p is no longer deemed to be at location l.
- *update*(*load-truck(p,t,l)*) = {}. There is no numeric update to be performed.

Other actions include *unload-truck* for unloading a package from truck, *board-truck* when a driver boards a truck, *disembark-truck* for disembarking the driver, and *walk* for displacing the driver on foot.

2.2 Plans and Plan Databases

In this section, we formally define a plan (and a plan database). Intuitively, a plan consists of a set of actions and constraints on the start and/or end times of actions. Due to space constraints, we use natural numbers to model time (formal calendars can be used with no difficulty). We use the variables $\mathsf{st}(\alpha(\boldsymbol{t}))$ and $\mathsf{et}(\alpha(\boldsymbol{t}))$, respectively, to denote the start and end times of an action.

Note that any durative action $\alpha(\boldsymbol{t})$ can be split into three simple action instances. $\alpha_{start}(\boldsymbol{t})$ describes what happens at the start of the durative action. $\alpha_{interval}(\boldsymbol{t})$ is a simple action describing the conditions that must hold during the duration of the action (no changes occur during execution). $\alpha_{end}(\boldsymbol{t})$ is a simple action describing changes at the end. We use $SIMPLE(\alpha(\boldsymbol{t}))$ to denote this set of three simple actions associated with $\alpha(\boldsymbol{t})$. For $\alpha(\boldsymbol{t})$ to be executable, the precondition of $\alpha_{start}(\boldsymbol{t})$ must hold when we start executing α and the precondition of $\alpha_{end}(\boldsymbol{t})$ must hold at just before we finish executing. During execution, the precondition of $\alpha_{interval}(\boldsymbol{t})$ must be true. We now state this formally:

Definition 7. *Suppose pw_i denotes the planworld at time i. An action instance $\alpha(\boldsymbol{t})$ is **executable** in a sequence of planworlds $[pw_1, \ldots, pw_k]$ if all of the following hold:*

- *$\alpha_{start}(\boldsymbol{t})$ is executable in pw_1;*
- *$\forall i, 1 < i < k$, $\alpha_{interval}(\boldsymbol{t})$ is executable in pw_i;*
- *$\alpha_{end}(\boldsymbol{t})$ is executable in pw_k;*

where $k = \mathsf{et}(\alpha(\boldsymbol{t}))$

Furthermore action instances $\alpha(\boldsymbol{t})$ and $\beta(\boldsymbol{z})$ are *mutually exclusive* an action in $SIMPLE(\alpha(\boldsymbol{t}))$ and an action in $SIMPLE(\beta(\boldsymbol{z}))$ are mutually exclusive and happen at the same time.

Informally speaking, a plan is a set of action instances that are pairwise mutually compatible with each other and that are executed in accordance with some temporal constraints. We define execution constraints below.

Definition 8. *An **execution constraint** for an action $\alpha(\boldsymbol{t})$ is an expression of the form $\mathsf{st}(\alpha(\boldsymbol{t})) = c$ or $\mathsf{et}(\alpha(\boldsymbol{t})) = c$ where c is a natural number.*

Definition 9. *A **plan** w.r.t. a (finite) set $\mathcal{A}$ of actions is a pair $\langle \mathcal{A}', C \rangle$ where $\mathcal{A}'$ is a set of action instances from $\mathcal{A}$ and C is a set of execution constraints w.r.t. actions in $\mathcal{A}'$.*[5]

A plan is definite *if for all actions in $\mathcal{A}'$, there are two execution constraints in C, one for constraining the start time and the other constraining the end time.*

Goals. In AI planning, a plan normally is generated to achieve some *goal* that is represented as a set of literals g. Though we do not define goals explicitly, there is no loss of generality because each goal g can be encoded in the plan as a special action whose pre-condition is g and whose effects are empty. The duration of this action specifies how long the goal conditions should be protected in the plan world after the completion of the plan.

We only consider definite plans in this paper, rather than allowing the start and end times of actions to vary.

[5] A minor problem here is how to handle plans in which the same action (e.g., *refuel(truck1)*) occurs more than once. An easy way to ensure that distinct action instances have a different names it to give each action instance an additional parameter called an *action identifier* that is different for each distinct action instance, e.g., *refuel(truck1,instance01)* and *refuel(truck1,instance02)*.

Example 3. Suppose we have packages p_1, p_2 at locations l_1, l_2 respectively. We have one truck t_1 at l_1 and a driver d in it. We want to deliver p_1 and p_2 to l_3. One possibility is to load p_1, then pick up p_2, and then go to the destinations. Here, $\langle\, \mathcal{A}', C \,\rangle$ is:

- $\mathcal{A}' = \{\ a_1 = \textit{load-truck}(p_1, t_1, l_1),\ a_2 = \textit{drive-truck}(t_1, l_1, l_2, d),\ a_3 = \textit{load-truck}(p_2, t_1, l_2),\ a_4 = \textit{drive-truck}(t_1, l_2, l_3, d),\ a_5 = \textit{unload-truck}(p_1, t_1, l_3),\ a_6 = \textit{unload-truck}(p_2, t_1, l_3)\}$
- $C = \{st(a_1) = 1, et(a_1) = 2, st(a_2) = 3, et(a_2) = 5, st(a_3) = 6, et(a_3) = 7, st(a_4) = 8, et(a_4) = 12, st(a_4) = 13, et(a_4) = 14, st(a_4) = 13, et(a_4) = 14\}$

C indicates an intuitive order for a package: load, drive, unload. Notice that two unload operations are performed concurrently.

Note that each action in a plan is an abstract realization of a physical process. For example, the action $\textit{drive}(t_1, l_1, l_2, d)$ is a syntactic representation of the physical action that a driver performs of driving from one place to another. Due to exogenous events, an action may not always succeed when carried out in the real world.

Definition 10. *A **plan database** is a 4-tuple $\langle \mathcal{PS}, pw, \texttt{plans}, \mathsf{now} \rangle$, where $\mathcal{PS}$ is a planspace, pw is the current planworld,* `plans` *is a finite set of plans and* now *is the current time.*

3 Consistency and Coherence of Plan DBs

Not all plan databases are consistent. For example, if we have only 50 gallons of fuel at a given location at some time T, and two different plans each plan to use 40 of those 50 gallons at that location at time T, then we would have an inconsistency. Coherence, on the other hand, intuitively requires that the plans be executable: all plans in the database must, for example, have preconditions that are valid w.r.t. the other plans in the database and the initial planworld. To formalize these notions, we first introduce the concept of future planworlds.

3.1 Future Planworlds

Throughout this section, we let $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \mathsf{now} \rangle$ be some arbitrary but fixed plan database. We use $S_{plans}(i)$ (resp. $E_{plans}(i)$) to denote the set of all actions in `plans` whose start (resp. end) time is i. $I_{plans}(i)$ is the set of the actions that start before time i and end after time i. The set of ***active actions*** at time i w.r.t. a given set *plans* of plans is defined as:

$$Active_{plans}(i) = \left(\bigcup_{\alpha \in S_{plans}(i)} \alpha_{start}\right) \cup \left(\bigcup_{\alpha \in E_{plans}(i)} \alpha_{end}\right) \cup \left(\bigcup_{\alpha \in I_{plans}(i)} \alpha_{interval}\right).$$

Suppose `plans` is given, the current time is i, and the plan world at time i is pw_i. What should the planworld at time t_{i+1} be, according to `plans`? We use $\mathcal{PW}^i_{plans}(R)$ to denote the extent of relation R at time i w.r.t. `plans`.

Definition 11 (future planworlds). *For all R,*

$$\mathcal{PW}^0_{plans}(R) = R$$
$$\mathcal{PW}^{i+1}_{plans}(R) = (\mathcal{PW}^i_{plans}(R) - Del_R(i, \texttt{plans})) \bigcup Add_R(i, \texttt{plans}),$$

where $Add_R(i, \texttt{plans})$ and $Del_R(i, \texttt{plans})$ are the set of all insertions and deletions, respectively to/from relation R by all actions in $S_{\texttt{plans}}(i) \cup E_{\texttt{plans}}(i)$. In the special case where R is a numeric relation, all tuples whose values are updated will be in $Del_R(i, \texttt{plans})$ and $Add_R(i, \texttt{plans})$ will contain the updated tuples with the new values, v'. If a numeric variable is updated at time i by a set of concurrent plan updates, then its new value will be computed as the old value plus the sum of all increases and decreases.

Assumptions. The above definition assumes that (i) when an action is successfully executed, its effects are incorporated into the planworld one time unit after the action's completion; (ii) all the actions in `plans` are successfully executable until information to the contrary becomes available; (iii) none of the actions are mutually exclusive with each other.

We now formally define the concept of consistency. Intuitively, consistency of a plan database requires that at all time points t, no two actions are mutually exclusive.

Definition 12. *Let P be a set of plans,* now *be the current time and e be the latest ending time in P. P is **consistent** if for every t, $\mathsf{now} \leq t \leq e$, $Active_P(t)$ does not contain any two simple actions that are mutually exclusive.*

The following algorithm can be used to check consistency of a set P of plans.

Algorithm $ConsistentPlans(P, \mathsf{now})$
  L = ordered time points either at or one time unit before an action starts or ends in P;
  while L is not empty **do**
    t = First member of L; $L = L$ - $\{t\}$;
    if $(\exists \alpha, \beta \in Active_P(t))$ α and β are mutually exclusive
      then return $false$;
  return $true$.

The reader can verify that the loop in this algorithm is executed at most $4n$ times, where n is the number of actions in P. Note that consistency of a plan database does not mean that the plan can be executed. To execute all the plans in a plan database, we need to ensure that the precondition of each action is true in the state (i.e. at the time) in which we want to execute it. The notion of coherence intuitively captures this concept.

Definition 13. *Suppose pw is the planworld at time* now *and P is a consistent set of plans. Suppose e is the latest ending time of any action in P. P is **coherent** iff for every $\mathsf{now} \leq t \leq e$ every simple action in $Active_P(t)$ is executable in $\bigcup_R \mathcal{PW}^t_P(R)$ where $pw = \bigcup_R \mathcal{PW}^{\mathsf{now}}_P(R)$.*

Clearly, we would always like a plan database to be both consistent (no conflicts) and coherent (executable). The following algorithm may be used to check for coherence.

Algorithm $CoherentPlans(P, \mathsf{now}, pw)$
L = ordered time points either at or one time unit before an action starts or ends in P;
while L is not empty **do**
t = First member of L; $L = L$ - $\{t\}$;
if $(\exists \alpha \in Active_P(t))pw \not\models pre(\alpha)$ **then return** $false$;
if $(\exists \alpha, \beta \in Active_P(t))$ α and β are mutually exclusive
then return $false$;
$pw = (pw - \bigcup_R Del_R(t, P)) \bigcup (\bigcup_R Add_R(t, P))$;
return $true$.

Goals. In AI planning, a plan normally is generated to achieve some *goal* that is represented as a set of lierals g. Though we do not define goals explicitly, there is no loss of generality because each goal g can be encoded in the plan as a special action whose pre-condition is g and whose effects are empty. The duration of this action specifies how long the goal conditions should be protected in the plan world after the completion of the plan.

Suppose we already know a given plan database is consistent (coherent), and we want to modify the set of plans in the plan database (but not the other components of the plan DB). The following two theorems provide sufficient conditions to check if the modified set of plans is consistent (coherent).

Theorem 1. *Suppose a plan database* $PLDB = \langle \mathcal{PS}, pw, \mathtt{plans}, \mathsf{now} \rangle$ *is consistent. Let* $PLDB' = \langle \mathcal{PS}, pw, \mathtt{plans}', \mathsf{now} \rangle$. $PLDB'$ *is consistent if*

- $Actions(\mathtt{plans}') \subseteq Actions(\mathtt{plans})$ *and*
- $Constraints(\mathtt{plans}') \subseteq Constraints(\mathtt{plans})$,

where $Actions(\mathtt{plans}')$ *is the set of all actions in all plans in* $\mathtt{plans}'$ *and* $Constraints(\mathtt{plans}')$ *is the set of all constraints in all plans in* $\mathtt{plans}'$.

Theorem 2. *We use the same notation as in theorem 1. Suppose a plan database* $PLDB = \langle \mathcal{PS}, pw, \mathtt{plans}, \mathsf{now} \rangle$ *is coherent and* $\mathtt{plans}'$ *satisfies the conditions in Theorem 1.* $PLDB'$ *is coherent if:*

1. $Cond(\mathtt{plans}') \cap Effects(\mathtt{plans} - \mathtt{plans}') \equiv \emptyset$, *or*
2. *All actions in* $\mathtt{plans}'$ *end before any action in* $\mathtt{plans} - \mathtt{plans}'$ *starts.*

Here $Cond(P)$ *is the set of preconditions of all actions in* P *and* $Effects(P)$ *is the set of all the effects of all actions in* P.

4 Plan Database Algebra

We now define a plan database algebra (PDA for short) to query plan databases. PDA contains selection, projection, union, intersection, and difference operators.

In addition, we introduce a *coherent selection* operator cs and a *coherent projection* operator cp which is used to ensure coherence properties. A new ***fast forward*** operator can be used to query the database about future states. Note that this is different from a temporal database where future temporal states are explicitly represented. In a plan database, all we are given explicitly is that various actions are scheduled to occur at various times, and need to reason about when these actions are performed and their effects in order to answer queries about future states. Just reading the database is not adequate.

4.1 Future Plan Databases

In order to achieve this goal, we first define the concept of *future plan databases*. Recall that in Section 3.1, we introduced "future planworlds". This definition assumed that the plan database was coherent.

However, this may not always be the case. Future plan databases describe the state of the plan database by *projecting* into the future. We assume we start with a consistent (but not necessarily coherent) database.

Definition 14. *Suppose* $\langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ *is a plan database and that the current time is* now. *The* ***future plan database*** $PossDB$ *at time* i *for* $i \geq \texttt{now}$ *is defined inductively as follows:*

1. *For* $i = \texttt{now}$*:* $PossDB^i(\langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle) = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ *and* $\texttt{plans}^i = \texttt{plans}$.
2. *For* $i > \texttt{now}$*: Suppose* $PossDB^{i-1}(\langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle) = \langle \mathcal{PS}, pw^{i-1}, \texttt{plans}^{i-1}, (i-1) \rangle$. *Then* $PossDB^i(\langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle) = \langle \mathcal{PS}, pw^i, \texttt{plans}^i, i \rangle$*, where:*
   (a) $pw^i = (pw^{i-1} - \bigcup_R Del_R(i-1, \texttt{plans}^{i-1})) \bigcup (\bigcup_R Add_R(i-1, \texttt{plans}^{i-1}))$.
   (b) $\texttt{plans}^i = \{\langle A, C \rangle \mid \langle A, C \rangle \in \texttt{plans}^{i-1}, (A \cap CannotStart) = \emptyset\}$.
   (c) $CannotStart = \{\alpha \mid \alpha_{sub} \in Active, sub \in \{end, start, interval\}, pw^i \not\models pre(\alpha_{sub})\}$.
   (d) $Active = Active_{plans^{i-1}}(i)$

The above definition inductively defines the plan database at time i by constructing it from the plan database at time $(i-1)$.

4.2 Selection Conditions

Before defining selection, we first need to define selection conditions. Suppose $\mathcal{PS}$ is some arbitrary but fixed planspace. As usual, we assume the existence of variables over domains of all attributes in the relations present in the planspace. In addition, we assume the existence of a set of variables $Z_1, Z_2, \ldots$ ranging over plans, a set $A_1, A_2, \ldots$ of variables ranging over actions, and a set $Y_1, Y_2, \ldots$ of variables ranging over tuples (of a planworld). A *plan term* is either a variable of any of the above three kinds, a constant of the appropriate kind, or of the form $V.a$ where V is a variable of the above kinds and a is an attribute of the

term denoted by V. If $V_1, \ldots, V_k$ are all plan terms then $\langle V_1, \ldots, V_k \rangle$ is a plan term denoting a tuple. Terms denoting actions and plans have special attributes `START` and `END` that correspond to the start and end time of actions and plans. In addition, terms denoting actions have a special `name` attribute. In the following definition, we assume the existence of some arbitrary but fixed planspace.

Definition 15. *Atomic selection conditions (ASCs) are inductively defined as follows:*

1. *If Y is a tuple term and R is a relation, then $Y \in R$ is an ASC.*
2. *If P is a plan term and A is an action term, then $A \in P$ is an ASC.*
3. *If t_1, t_2 are terms of the same type, then $pt_1 = pt_2$ is an ASC.*
4. *If t_1, t_2 are terms of the same type and the type has an associated linear ordering $\leq$, then $t_1 \underline{op}\, t_2$ is an ARC, where $\underline{op} \in \{\leq, <, >, \geq, <>\}$.*

Definition 16. *A* simple *plan database condition (simple PDC) is inductively defined as: (i) every ASC is a simple PDC, and (ii) if X_1, X_2 are simple PDCs, then so are $(X_1 \wedge X_2)$ and $(X_1 \vee X_2)$.*

If X is a simple plan database condition and I is either a variable (over integers) or an integer, then the expression $[I] : X$ is a plan database condition (PDC).

The condition $[I] : X$ holds if the condition X evaluates to true at time I in the underlying plan database. If I is a constant, then the PDC is called a *time bounded expression*. Otherwise, we say that the PDC is an *unbounded expression*.

Definition 17 (satisfaction). Let X be a ground simple PDC and $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ be a plan database. The satisfaction of all atomic selection conditions by a $PLDB$ is defined in the obvious way. In addition, if p is a plan term, then $PLDB$ satisfies $p.\texttt{END}\ \underline{op}\ c$ if for all actions $\alpha \in p$, $\texttt{et}(\alpha) \leq \texttt{now}$, $z = max\{e \mid \texttt{et}(\alpha) = e, \alpha \in p\}$ and $z\ \underline{op}\ c$ holds. Similarly, $PLDB$ satisfies $p.\texttt{START}\ \underline{op}\ c$ if there is an action $\alpha \in p$ such that $\texttt{st}(\alpha) < \texttt{now}$, $z = min\{s \mid \texttt{st}(\alpha) = s, s \leq \texttt{now}, \alpha \in p\}$ and $z\ \underline{op}\ c$ holds. If α is an action term and p is a plan, then $PLDB$ satisfies $\alpha \in p$ if $p \in \texttt{plans}$, $p = \langle \mathcal{A}', \mathcal{C} \rangle$ and $\alpha \in \mathcal{A}'$. If a is an action term in plan p then $PLDB$ satisfies $\alpha.\texttt{START}\ \underline{op}\ c$ iff $\texttt{st}(\alpha) = s$, $s \leq \texttt{now}$ and $s\ \underline{op}\ c$ holds. Similarly, $PLDB$ satisfies $\alpha.\texttt{END}\ \underline{op}\ c$ iff $\texttt{et}(\alpha) = e, e < \texttt{now}$ and $e\ \underline{op}\ c$ holds. If R is a relation name then $PLDB$ satisfies $Y \in R$ succeeds for a tuple term Y iff tuple Y is in the relation R according to pw. Suppose i is an integer, $PLDB$ satisfies $[I] : X$ if and only if X is true in $PossDB^i(PLDB)$. If $[I] : X$ is a non-ground PDC then, $PLDB$ satisfies $[I] : X$ if there exists a ground instance $[I] : X\gamma$ such that $PLDB \models [I] : X\gamma$. If $[I] : X$ is an unbounded (i.e. I is a variable) PDC then $PLDB$ satisfies $[I] : X$ iff there exists an integer i such that $PLDB \models [I] : X$.

As usual, we use the symbol $\models$ to denote satisfaction.

Example 4. To find all actions that finish before time 20, we can write $(A.\texttt{END} \leq 20)$. To find all plans that will finish successfully, we can write $[I] : (Z.\texttt{END} \leq I)$. In this expression, we want to find a time instance I where all plans in the database at time I finish successfully (before time I).

The following algorithm finds all plans that successfully end before time i. The algorithm is useful if a plan database is not coherent. If the plan database is coherent, we know for certain that all the plans in the plan database will succeed unless an exogenous real world event intervenes (which would lead to a database update).

Algorithm PlansSuccessfullyEnd(PDB, i)
$Ans = \emptyset$;
$\langle \mathcal{PS}, pw^i, \texttt{plans}^i, i \rangle = PossDB^i(PDB)$;
while $\texttt{plans}^i \neq \emptyset$ **do**
Select $\langle \mathcal{A}, \mathcal{C} \rangle \in \texttt{plans}^i$; $\texttt{plans}^i = \texttt{plans}^i - \{\langle \mathcal{A}, \mathcal{C} \rangle\}$;
if there is no $\alpha \in \mathcal{A}$ such that $\texttt{et}(\alpha) > i$ **then** $Ans = Ans \cup \{\langle \mathcal{A}, \mathcal{C} \rangle\}$
return Ans.

It is easy to see that that above algorithm can be executed in time proportional to the number of plans in the plan database.

4.3 Selection

The selection operation finds all plans (and their associated information) that satisfy a specific condition.

It is important to note that selection may not preserve coherence. For instance, suppose we have a database containing five plans $p_1, \ldots, p_5$ and suppose p_1, p_2, p_3 satisfy the selection condition. Then these are the plans that the user wants selected. However, p_2 may have actions in it that depend upon the prior execution of p_4 (otherwise the preconditions of p_2 may not be true). Coherence would require that we add p_4 to the answer as well. For this reason, we define two versions of the selection operator - ordinary selection which does not necessarily guarantee coherence, and coherent selection which would add a minimal number of extra plans to guarantee coherence.

Definition 18. Suppose $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ is a plan database and $[I] : X$ is a PDC involving a plan variable Z. The *plan selection* operation, denoted by $\sigma_{[I]:X}\, PLDB(Z) = \langle \mathcal{PS}, pw, \texttt{plans}', \texttt{now} \rangle$, is computed as $\texttt{plans}' = \{\langle \mathcal{A}, \mathcal{C} \rangle \mid \langle \mathcal{A}, \mathcal{C} \rangle \in sol(Z)\}$, where

$$sol(Z) = \{\langle \mathcal{A}, \mathcal{C} \rangle \in \texttt{plans} \mid PLDB \models [I] : X/\{Z = \langle \mathcal{A}, \mathcal{C} \rangle\} \text{ and } \nexists I' < I \text{ such that } PLDB \models [I'] : X/\{Z = \langle \mathcal{A}, \mathcal{C} \rangle\}\}.$$

Proposition 1. *If $PLDB$ is consistent, then according to Theorem 1, $\sigma_{[I]:X}\, PLDB(Z)$ is also consistent. If $PLDB$ is coherent and $\sigma_{[I]:X}\, PLDB(Z)$ satisfies either of the conditions in Theorem 2, then $\sigma_{[I]:X}\, PLDB(Z)$ is also coherent.*

Example 5. Suppose we want to retrieve all plans in which a certain driver, say Paul, drives the truck. We can write the following plan selection query: $\sigma_{[I]:X}\, PLDB(Z)$ where $X = (A = \textit{drive-truck}(\_, \_, \_, \textit{paul}) \wedge A \in Z)$.

Suppose the initial plan database $PLDB$, contains the following plans;

- $P_1 = \langle\{a_1 = \textit{board-truck}(paul, t_1, c_1), a_2 = \textit{board-truck}(paul, t_1, c_2),$ $a_3 = \textit{board-truck}(ted, t_2, c_3)\}, \{\ \mathsf{st}(a_1) = 1,\ \mathsf{et}(a_1) = 3,\ \mathsf{st}(a_2) = 9,\ \mathsf{et}(a_2) = 11,$ $\mathsf{st}(a_3) = 1,\ \mathsf{et}(a_3) = 3\}\rangle$
- $P_2 = \langle\{a_4 = \textit{drive-truck}(t_1, c_1, c_2, paul), a_5 = \textit{drive-truck}(t_2, c_1, c_2, ted)\},$ $\{\mathsf{st}(a_4) = 4, \mathsf{et}(a_4) = 8, \mathsf{st}(a_5) = 6, \mathsf{et}(a_2) = 11\}\rangle$
- $P_3 = \langle\{a_6 = \textit{walk}(paul, c_2, c_3)\}, \{\mathsf{st}(a_6) = 12, \mathsf{et}(a_6) = 16\}\rangle$

and the current time is 0. In this case, the above query returns only P_2. However the plan database which contains just P_2 is not coherent at time 0 because at time 4, Paul will not be in truck t_1 which is one of the conditions of action a_4. The coherent selection operation will fix this.

Example 6. Suppose we want to retrieve all plans in which the same driver has to deliver items to at least three different places. We can write the following plan selection expression: $\sigma_{[I]:X} PLDB(Z)$ where $X = (A1 = \textit{drive-truck}(\_, \_, L1, D) \wedge A2 = drive-truck(\_, \_, L2, D) \wedge A3 = drive-truck(\_, \_, L3, D) \wedge A1 \in Z \wedge A2 \in Z, \wedge A3 \in Z \wedge L1 \neq L2 \wedge L1 \neq L3 \wedge L2 \neq L3)$.

4.4 Coherent Selection

Selection is guaranteed to preserve consistency, but not coherence. Fortunately, we can restore coherence by using the algorithm `ClosePlans` below. The algorithm invokes a subroutine called `SupportivePlans`(P, F, t). For every action $\alpha \in F$, `SupportivePlans` nondeterministically[6] selects a plan in P that contains an action β with an effect e which establishes the precondition of α. It also ensures that $\mathsf{st}(\beta)$ (resp. $\mathsf{et}(\beta)$) is less than t, if e is an effect of β_{start} (resp. β_{end}). `SupportivePlans` returns the set of selected plans. The algorithm is guaranteed to terminate if the input plan DB `plans` is coherent wrt pw and `now`.

Algorithm `ClosePlans`$(\mathcal{PS}, pw, \texttt{plans}, \texttt{now}, \texttt{plans}')$
  $last$ = Latest ending time in `plans'`;
  $t = \texttt{now}$; $pw_t = pw$;
  while $t \leq last$ do
    $A \equiv Active_{plans'}(t)$
    if $A \equiv \emptyset$ then
      $pw_{t+1} = pw_t$;t= t+1;
    else if $\forall \alpha \in A, pw_t \models pre(\alpha)$ **then**
      $pw_{t+1} = pw_t - Del_{plans'}(t) + Add_{plans'}(t)$;
      $t = t + 1$;
    else
      $F \equiv \{\alpha | \alpha \in A, pw_t \not\models pre(\alpha)\}$;
      $P =$ `SupportivePlans`$(\texttt{plans} - \texttt{plans}', F, t)$;
      $t =$ Earliest start time of actions in P;

[6] Note that any nondeterministic operation can be made deterministic by defining a linear order on all choices and simply choosing the choice that is minimal w.r.t. the linear order. Due to space limitations, we do not pursue this option here.

$\texttt{plans}' \equiv \texttt{plans}' \cup P$
$last$ = Latest ending time in plans';
return plans′

We now define the coherent selection operator cs that guarantees coherence.

Definition 19. Suppose $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ is a plan database and $[I] : X$ is a PDC involving a plan variable Z. The ***coherent selection*** operation, denoted by $\texttt{cs}_{[I]:X}\ PLDB(Z) = \langle \mathcal{PS}, pw, \texttt{plans}*, \texttt{now} \rangle$, is given by:

- $\langle \mathcal{PS}, pw, \texttt{plans}', \texttt{now} \rangle = \sigma_{[I]:X}\ PLDB(Z)$
- $\texttt{plans}* = \texttt{ClosePlans}(\mathcal{PS}, pw, \texttt{plans}, \texttt{now}, \texttt{plans}')$

Example 7. Let us return to Example 5, where we want to select all plans in which Paul drives. The *coherent selection* operation would return the plan DB containing both P_2 and P_1 which will be coherent.

4.5 Projection

The projection operation selects plans which contain actions that satisfy a specific condition. For a plan, only the actions that satisfy the conditions are kept, the others are removed from the plan. As in the case of *selection*, the coherence property may be violated after a projection. Later, we will introduce a coherence preserving projection operation that establishes coherence by reinserting some actions and/or plans removed during projection to reestablish the necessary coherence property.

Definition 20. Suppose $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ is a plan database and $[I] : X$ is a PDC involving a variable A denoting an action. The ***action projection*** operation, denoted $\Pi_C\ PLDB(A) = \langle \mathcal{PS}, pw, \texttt{plans}', \texttt{now} \rangle$, is defined as:

- $\texttt{plans}' = \{\langle \mathcal{A}^*, \mathcal{C}^* \rangle \mid \langle \mathcal{A}', \mathcal{C} \rangle \in \texttt{plans}, \mathcal{A}^* = \{\alpha \mid \alpha \in \mathcal{A}' \text{ and } \alpha \in sol(A)\}, \mathcal{C}^* = rest(\mathcal{C}, \mathcal{A}^*)\}$,

where

- $sol(A) = \{\alpha \mid \langle \mathcal{A}', \mathcal{C} \rangle \in \texttt{plans}$ and $PLDB \models [I] : X/\{A = \alpha\}$ and $\nexists I' < I$ such that $PLDB \models [I'] : X/\{A = \alpha\}\} \cup \{\alpha \mid \langle \mathcal{A}', \mathcal{C} \rangle \in \texttt{plans}$ and $\texttt{st}(\alpha) \leq \texttt{now}\}$;
- $rest(\mathcal{C}, \mathcal{A}^*)\} = \bigcup_{\alpha \in \mathcal{A}^*} \{\text{all execution constraints for } \alpha \text{ in } \mathcal{C}\}$.

We note that the action projection will return actions that satisfy the given conditions and started already.

Proposition 2. *If $PLDB$ is consistent then so is $\Pi_{[I]:X}\ PLDB(A)$. If $PLDB$ is coherent and $\Pi_{[I]:X}\ PLDB(A)$, satisfies either of the conditions in Theorem 2, then $\Pi_{[I]:X}\ PLDB(A)$, is also coherent.*

Example 8. Suppose we want to retrieve plans only consisting of *drive-truck* actions. Specifically, we only want to keep those actions for which there exists, in their own plan, another delivery that has the same driver, and the second delivery happens after x time units. We can use the following plan projection query: $\Pi_{[I]:X}\ PLDB(A)$ where $X = (A1 = \textit{drive-truck}(\_, \_, \_, D) \wedge A2 =$ drive-truck$(_, _, _, D) \wedge\ A1 \neq A2 \wedge et(A1) - et(A2) = x, \wedge(A = A1 \vee A = A2))$.

Example 9. Consider once more the selection query in Example 7. A projection operation with the same condition on the same plan database will yield a plan database with the following single plan in it: $P_2' = \langle\{a_4 = \textit{drive}(t_1, c_1, c_2, \textit{paul})\}, \{\mathsf{st}(a_4) = 4, \mathsf{et}(a_4) = 8\}\rangle$. As explained earlier, a_4 will fail because there is no driver in $\textit{truck}_1$.

4.6 Coherent Projection

We now define a ***closed plan projection*** operator `cp` similar to the coherent selection operator. It will return the plans with actions that satisfy the selection criteria as well as the other actions needed to make the projected set of plans coherent.

Definition 21. Suppose $PLDB = \langle\, \mathcal{PS}, pw, \mathtt{plans}, \mathsf{now}\,\rangle$ is a plan database and $[I] : X$ is a PRC. The ***closed plan projection operation***, denoted $\mathsf{cp}_{[I]:X}\ PLDB(A) = \langle\, \mathcal{PS}, pw, \mathtt{plans}*, \mathsf{now}\,\rangle$, is given by:

- $\langle\, \mathcal{PS}, pw, \mathtt{plans}', \mathsf{now}\,\rangle = \pi_{[I]:X}\ PLDB(A, Z)$
- $\mathtt{plans}* = \mathtt{CloseActions}(\mathcal{PS}, pw, \mathtt{plans}, \mathsf{now}, \mathtt{plans}')$

The definition of closed projection requires a `CloseActions` procedure which is a slight variation of the `ClosePlans` algorithm. Instead of calling `SupportivePlans`, it calls a `SupportiveActions` which is a slight variant of `SupportivePlans`: basically this procedure returns plans restricted to the supporting actions. The following example shows the use of the coherent projection operator.

Example 10. Let us return to the case of Example 9 and use coherent projection instead of projection. The resulting plan DB contains two plans:

- $P_1' = \langle\{a_1 = \textit{board-truck}(\textit{paul}, t_1, c_1)\}, \{\mathsf{st}(a_1) = 1, \mathsf{et}(a_1) = 3\}\rangle$,
- $P_2' = \langle\{a_4 = \textit{drive-truck}(t_1, c_1, c_2, \textit{paul})\}, \{\mathsf{st}(a_4) = 4, \mathsf{et}(a_4) = 8\}\rangle$.

This database is coherent. Notice the difference between number of actions added by coherent selection and coherent projection. In the first case, the total number of actions added into the plan database is three whereas in the second case it is only one. This is because coherent selection includes a plan with all its actions, whereas coherent projection only includes the necessary actions.

4.7 Fast Forward

In this section we define the ***fast-forward*** operator which returns future states of a plan database that satisfy various PDC conditions. The fast forward operation can be thought of as a projection operation into the future. Note however, that unlike a temporal database, we cannot look just at the relational state - we must also see how this relational state changes over time as the various actions in the plan database are executed according to the given schedule.

Definition 22. Suppose $PLDB = \langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ is a plan database and $[I] : X$ is a PDC. The ***fast forward*** of database $PLDB$ with respect to $[I] : X$, is $\Gamma_{[I]:X}(PLDB) = PossDB^I(PLDB)$, where I is the smallest integer such that $PLDB \models [I] : X$ if such an I exists. If no such I exists, then $\Gamma_{[I]:X}(PLDB)$ is undefined.

Proposition 3. *If $PLDB$ is consistent/coherent and $\Gamma_{[I]:X}(PLDB)$ is defined then $\Gamma_{[I]:X}(PLDB)$ is also consistent/coherent.*

4.8 Union, Intersection, Difference

In this section we describe the *union, difference* and *intersection* operations for plan databases. We first define the notion of *union compatibility* which simply states that the data in two plan worlds must have same values for the same numeric variables. The reason for this is that if one plan world says there 10 gallons of fuel, and another plan world says there are 20, then the union yields something claiming there are both 10 and 20 gallons of fuel which is problematic. Unlike intersection and difference, union does not necessarily preserve consistency even when the plan databases involved are union compatible. However, in Theorem 3 below, we state some conditions that are sufficient to preserve consistency. Two databases $\langle \mathcal{PS}, pw, \texttt{plans}, \texttt{now} \rangle$ and $\langle \mathcal{PS}, pw', \texttt{plans}', \texttt{now} \rangle$ are *union compatible* if every numeric variable f that is both in pw and pw' has the same value in both plan worlds.

Definition 23. Let $PLDB_1 = \langle \mathcal{PS}, pw_1, \texttt{plans}_1, \texttt{now} \rangle$ and $PLDB_2 = \langle \mathcal{PS}, pw_2, \texttt{plans}_2, \texttt{now} \rangle$ be two union compatible plan databases. Suppose the plans in $\texttt{plans}_2$ are renamed so that there are no plans with the same identifier in both databases. Then, the ***union*** of $PLDB_1, PLDB_2$, denoted $PLDB_1 \cup PLDB_2$ is given by

$$PLDB_1 \cup PLDB_2 = \langle \mathcal{PS}, pw_1 \cup pw_2, \texttt{plans}_1 \cup \texttt{plans}_2, \texttt{now} \rangle.$$

The following theorem states conditions guaranteeing consistency of the union of two union-compatible plan databases.

Theorem 3. *Suppose $PLDB_1$ and $PLDB_2$ are consistent. $PLDB_1 \cup PLDB_2$ is consistent if $\forall(\alpha \in Actions(\texttt{plans}_1), \beta \in Actions(\texttt{plans}_2))$ either of the following holds:*

1. $(Cond(\alpha) \cup Effects(\alpha)) \bigcap (Cond(\beta) \cup Effects(\beta)) = \emptyset$;

2. $\mathsf{st}(\alpha) > \mathsf{et}(\beta)$ *or* $\mathsf{st}(\beta) > \mathsf{et}(\alpha)$*;*

where $Actions(\texttt{plans})$ *is the set of all actions of all plans in* `plans`*,* $Cond(\alpha)$ *is the set of conditions of* α *and* $Effects(\alpha)$ *is the set of all effects of* α*.*

Theorem 3 is intuitive. If any two actions that access the same atoms do not overlap in time, then they cannot be mutually exclusive because none of their simple actions will happen at the same time. Any other two actions with overlapping executions will not be mutually exclusive since they don't modify the truth values of the same atoms.

The intersection and difference between two plan databases can be defined analogously, but we omit the definitions due to lack of space. Note that union, intersection, and difference may not be coherent even if the input plan DBs are coherent. We can define *coherent* union, intersection and difference operators in a manner similar to the coherent selection and projection operators.

5 Related Work

To date, there has been no other work on developing plan databases. We are aware of no formal query language for querying plans analogous to the relational algebra or relational calculus. However, there are two related areas: case based planning and temporal databases.

The goal of case based planning [6] is to store plans in a "case base" so that when we need to solve a new planning problem, we can examine the case base and identify similar plans that can be modified to solve the new planning problem. Our goal in this paper is very different. We are interested in *querying large databases of plans* so that different applications can perform their tasks. Such applications involve logistics where a transportation company may wish to examine plans and schedules to determine how to allocate resources (using operations research methods perhaps) as well as to analyze traffic, as well as air traffic control where we wish to identify when and where aircraft will be in the future so as to avoid potential mishaps. Some important aspects of our framework and consistency and coherence of the database. In contrast, case based planners do not require consistency nor coherence because the case base is not a set of plans being executed; rather, it is a library and the queries to this library concentrate on similarity issues.

There are also connections between our work and work in temporal databases [9, 2]. In temporal relational databases, we have two kinds of time: transaction time and valid time. Transaction time databases store information about when a given tuple was inserted into a relation, when updates were made, etc. Therefore, such databases deal with past events, not future events. In addition, they only deal with actions that affect the database. In contrast, in planning, we deal with actions that are intended to be executed in the future, these actions have an effect on the real world, and these effects are represented in the database by making updates to the database at appropriate future time instances. This

involves notions like coherence and consistency that are not relevant for transaction time (notions of consistency associated with database locking are very different). Valid time usually associates with an ordinary relational tuple, either a single time stamp, or a time interval. These denote the time when an event is true (or a time interval throughout which the event is true). Even though the start and end times of actions can be stored in a temporal database, temporal databases do not reason about the effects of these actions and allow queries that require reasoning about such effects.

There are also a few pieces of work [5, 3] involving non-deterministic time, in which one can make statements of the form "An event is valid at some time point in a given interval" (as compared to being true throughout the interval). Consistency here can be important [10, 7]. Users might be interested in temporal queries such as "Find all events starting after some time t or after completion of some other event e." Processing such queries requires checking consistency of temporal constraints. In such temporal constraint databases and query languages, the temporal constraints used can be much more expressive than those used in our model. However, the purposes are very different. These works discuss the occurrence of events at time points in the future, but not about the fact that these events could be actions that have an impact on the world. As a consequence, they do not model the fact that their events can trigger updates to the database. Hence, there is no need in their frameworks for concepts like consistency, coherence, and closure introduced here, and our definitions of the algebraic operations are correspondingly different.

6 Conclusions

Many agencies and corporations store complex plans—ranging from production plans to transportation schedules to financial plans— composed of hundreds of "interlinked" plan elements. Such applications require not only that plans be created automatically, but also that they be stored in an appropriate data model, and that they be monitored and tracked during as they (i.e. the plans) are executed. To date, most work on plans has focused on the *creation* of plans.

In this paper, we propose a *data model* for storing plans so that plans may be monitored and tracked. We propose the concept of a *plan database* and provide algebraic operations to query such databases. These algebraic operations extend the classical relational operations of selection, projection, join, etc. In addition, we provide algorithms to update sets of plans as new plans need to be added to the database, and as old plans are executed (and either adhere or do not adhere to their intended schedules).

Much future work remains to be done on plan databases, and this paper merely represents a first step. Topics for future study include scalable disk-based index structures to query plan databases, cost models for plan algebra operations, equivalences of queries in plan databases, and optimizing queries to plan databases.

7 Acknowledgments

This work was supported in part by the following grants, contracts, and awards: Air Force Research Laboratory F30602-00-2-0505, Army Research Laboratory DAAL0197K0135, DAAL0197K0135 and DAAD190320026, Naval Research Laboratory N00173021G005, the CTA on Advanced Decision Architectures, ARO contracts DAAD190010484 and DAAD190310202, DARPA/RL contract number F306029910552, NSF grants IIS0222914 and IIS0329851 and the University of Maryland General Research Board. Opinions expressed in this paper are those of authors and do not necessarily reflect opinion of the funders.

References

1. P. Brucker. *Scheduling Algorithms*. Springer-Verlag, New York, 1995.
2. J. Chomicki. Temporal query languages: a survey. In *Temporal Logic: ICTL'94*, volume 827, pages 506–534. Springer-Verlag, 1994.
3. C. E. Dyreson and R. T. Snodgrass. Supporting valid-time indeterminacy. *ACM Transactions on Database Systems*, 23(1):1–57, 1998.
4. M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains, 2002. http://www.dur.ac.uk/d.p.long/pddl2.ps.gz.
5. S. Gadia, S. Nair, and Y. Peon. *Incomplete Information in Relational Temporal Databases*. Vancouver, 1992.
6. K. J. Hammond. Case-Based Planning: Viewing planning as a memory task (Academic Press, San Diego, CA, 1989).
7. M. Koubarakis. Database models for infinite and indefinite temporal information. *Information Systems*, 19(2):141–173, 1994.
8. D.-T. Peng, K. G. Shin, and T. F. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *Software Engineering*, 23(12):745–758, 1997.
9. R. Snodgrass. The temporal query language tquel. *ACM Transactions on Database Systems (TODS)*, 12(2):247–298, 1987.
10. P. T. V. Brusoni, L. Console and B. Pernici. Qualitative and quantitative temporal constraints and relational databases:theory, architecture and applications. *IEEE TKDE*, 11(6):948–968, 1999.