Compiling Plan Operators from Domains Expressed in Qualitative Process Theory

John C. Hogge
Qualitative Reasoning Group
Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

The study of Qualitative Physics has concentrated on expressing qualitatively how the physical world behaves. Qualitative Physics systems accept partial descriptions of the world and output the possible changes that can occur. These systems currently assume that the world is left untouched by human or robot agents, limiting them to certain types of problem solving. For instance, a state-of-the-art qualitative physics system can diagnose faulty electrical circuits but can not construct plans to rewire circuits to change their behavior. This paper describes an approach to planning in physical domains and a working implementation which integrates Forbus' Qualitative Process Engine (QPE) with a temporal interval-based planner. The approach involves compiling QPE expressions describing a physical domain into a set of operators and rules of the planner. The planner can then construct plans involving processes, existence of individuals, and changes in quantities. We describe how the compilation is performed, the types derivable plans, and current limitations in our approach.

1 Introduction

The study of Qualitative Physics has concentrated on expressing qualitatively how the physical world behaves. Qualitative Physics systems accept partial descriptions of the world and output the possible changes that can occur. These systems currently assume that the world is left untouched by human or robot agents, limiting them to certain types of problem solving. For instance, a state-of-the-art qualitative physics system can diagnose faulty electrical circuits but can not construct plans to rewire circuits to change their behavior.

This paper describes an approach to planning in physical domains and a working implementation which integrates a particular qualitative physics system, the Qualitative Process Engine (QPE) [Forbus, 86], with a planner (TPLAN) based on [Allen and Koomen, 83]. The implementation, called the Operator Compiler, accepts QPE expressions describing a physical domain and compiles a set of TPLAN operators for achieving goals that require processes to occur. For instance, given the definition for a liquid flow process, the Operator Compiler outputs an operator for creating liquid flows. This operator can solve goals matching the effects of liquid flow, such as the increased amount of liquid in a container.

The Operator Compiler could prove useful in applications, due to its integration two powerful systems. QPE envisions what can happen in the world from various states, while TPLAN plans changes in the world. Furthermore, once a domain physics has been constructed and debugged using QPE, adding planning capabilities requires little work. The user can design the physics without worrying about the task of planning. Formalizing the rest of the domain (such as an agent's possible actions) then requires some use of the physics vocabulary (to relate actions to processes, for instance).

Sections 2 and 3 describe aspects of QPE and TPLAN relevant to understanding what the Operator Compiler does. The Operator Compiler is presented in section 4, followed by discussion in section 5.

2 Introduction to QPE

QPE is an implementation of Qualitative Process Theory (QPT) [Forbus, 84]. We use the term QPT to refer to the language with which one models physical processes, ignoring other aspects such as the deductions it sanctions. Examples of processes are liquid flow, heat flow, and boiling. QPT provides a syntax for describing individuals in a domain and for expressing how processes become active and how they affect individuals. Examples of individuals are containers, fluid paths, liquid sources, quantities of individuals, and processes. Like other qualitative physics theories, QPT models quantities as a partial order and uses symbolic, rather than numeric values. Thus, inequalities such as `(GREATER-THAN quantity1 quantity2)` are meaningful, but measured amounts are irrelevant. Quantities have two components: amount (signified by A) and derivative (signified by D).

QPT defines a process as a collection of five components: individuals involved in the process, preconditions (outside of QPT's knowledge) on the process, quantity conditions (inequalities), relations asserted during the process, and influences the process puts on quantities. Figure 1 shows a typical process definition.

3 Introduction to TPLAN

TPLAN is an implementation of the temporal interval-based planner described in [Allen and Koomen, 83]. TPLAN keeps a database of facts qualified by intervals over which they hold. The planner runs on top of a time logic described in [Allen, 83], which maintains temporal relationships between intervals. Table 1 shows the possible values a relation can have.

TPLAN adopts the following syntactic conventions:

1. Intervals are denoted by symbols starting with "$".
2. Variables are denoted by symbols starting with "?".

```
Process: (LIQUID-FLOW ?src-can ?dst-can ?liq)
Individuals:  (CONTAINER ?src-can)
              (CONTAINER ?dst-can)
              (CONTAINED-LIQUID (CL ?liq ?src-can))
              (FLUID-PATH (FP ?src-can ?dst-can))
Preconditions:  (VALVE-OPEN (FP ?src-can ?dst-can))
Quantity Conditions:
  (GREATER-THAN (A (PRESSURE ?src-can))
                (A (PRESSURE ?dst-can)))
Relations:  (QUANTITY FLOW-RATE)
            (Q= FLOW-RATE (- (PRESSURE ?src-can)
                             (PRESSURE ?dst-can)))
Influences:  (I+ (AMOUNT-OF (CL ?liq ?dst-can))
                 (A FLOW-RATE))
             (I- (AMOUNT-OF (CL ?liq ?src-can))
                 (A FLOW-RATE))
```

Figure 1: Definition of Process Liquid Flow

Table 1: Seven Possible Values of Interval Relations and their Inverses

Value	Description	Inverse	Description
:<	before	:>	after
:M	meets	:MI	met by
:O	overlaps	:OI	overlapped by
:S	starts	:SI	started by
:F	finishes	:FI	finished by
:D	during	:DI	encloses
:=	equals	:=	equals

3. The temporal relation between two intervals is expressed as a disjunction and written as a list. For instance, (:< :>) means "is before or after."

4. Facts are paired with the interval over which they hold. Thus, `(ON A TABLE) $INTERVAL1` denotes a fact `(ON A TABLE)` which holds over `$INTERVAL1`.

5. Two facts paired with the same interval is equivalent to assigning them separate intervals constrained to be (:=).

These syntactic conventions are used in operator and rule definitions. We describe each in turn.

3.1 Operators

An operator defines an action the agent can perform to change the world. TPLAN adopts the model of action presented in [Allen and Koomen, 83], with temporally qualified patterns describing operator preconditions and effects. For instance, Figure 2 defines an operator PICKUP which can be applied to an object if it is clear and resting on something. PICKUP's effects clear the object's old location. The constraints field constrains the temporal relations among facts unifying with the preconditions and effects.

3.2 Rules

Rules model temporal laws of the domain. TPLAN uses rules as backward chaining operators for solving goals, as well as forward

```
OPERATOR pickup
  PRECONDITIONS: (clear ?object) $clear-object
                 (on ?object ?surface) $on
  EFFECTS: (pickup ?object ?surface) $pickup
           (clear ?surface) $clear-surface
  CONSTRAINTS: $clear-object (:M) $pickup
               $on (:O) $pickup
               $on (:M) $clear-surface
```

Figure 2: Definition of Operator PICKUP

chaining, temporally constrained inference rules. Thus, if we express QPT processes as rules, we can both infer processes and plan for their occurence.

```
RULE
  Antecedents: (ON ?x ?y) $on-xy
               (ON ?y ?z) $on-yz
  Temporal Conditions:
    Exists (INTERSECTION $on-xy $on-yz)
      called $intersection
  Consequents: (OVER ?x ?z) $intersection
  Consequent Constraints:
```

Figure 3: A Temporally Qualified Inference Rule

Rule definitions are similar to operator definitions, with antecedents behaving like operator preconditions, consequents behaving like operator effects, and consequent constraints behaving like operator constraints. The additional field, temporal conditions, places preconditions on the temporal relations among facts matching the antecedents. Our time logic supports temporal intersections, allowing us to inhibit a rule until antecedents are known to intersect (meaning their relation is a subset of (:S :SI :F :FI :D :DI :O :OI :=)) and to assert consequents over their intersection. Figure 3 demonstrates these features. Given (ON A B) and (ON B C) whose intervals intersect, (OVER A C) is asserted over their intersection.

4 The Operator Compiler

Qualitative physics systems reason about situations containing fixed sets of individuals. For instance, given a pot containing water resting on a burner, QPE can envision what might happen depending on the relative temperatures of the water and the burner. However, QPE can not envision what might happen if we move the pot off of the burner at some point unless we attempt to model agent actions in process definitions. While QPE does support such modeling (implemented as additional assumptions), the support is temporally weak and multiplies the size of the envisionment of possible world states by the number of consistent action combinations.

The Operator Compiler is an attempt to avoid this combinatorics by modeling actions within TPLAN, whose search through possible states is more constrained than QPE's envisionment. The Operator Compiler approach involves three steps:

1. The user models in QPT the individuals and physical pro-

cesses of the domain.

2. The Operator Compiler analyzes the model of step 1 and compiles operators and rules for constructing plans involving processes and individuals.

3. The user models in TPLAN the actions an agent can perform in the domain and the temporal laws of the domain. These actions must be made relevant to the output of step 2.

For example, assume we want to model a kitchen and generate plans involving liquid flows. In step 1 we model a liquid flow process and individuals such as containers, liquid sources, and valved fluid paths. In step 2, the Operator Compiler would output operators and rules which include the means by which a liquid flow is initiated. In step 3 we construct TPLAN rules that describe a simple Blocks World geometry for the kitchen and relate the geometry to the process descriptions. For instance, one rule establishes an exterior fluid path from faucets to any container underneath it. We then model actions relevant to the geometry and the physics. For instance, one geometric operator would permit us to move objects. Other operators would allow us to modify certain quantities in the physics, such as the valve position of faucets.

The vocabulary produced by these three steps can now be used to solve specific planning scenarios. For example, we could assert container POT1 initially on the counter, liquid source FAUCET having a non-zero amount of water, and liquid drain SINK underneath FAUCET and have TPLAN solve any of the following problems:

1. Increase the amount of water in POT1

2. Increase the pressure in POT1

3. Fill the sink with water

These problems may seem trivial. However, our running examples require complex plans such as the one traced in Figure 4, since our QPT domain model is detailed.

Figure 4 shows examples of the backward-chaining use of rules generated by the Operator Compiler. One rule is used to initiate a liquid flow to solve the initial goal. Later, a rule is used to infer that the faucet is a container if it is a liquid source. Such rules are simple to construct from QPT expressions because of QPT's notion of causality. Since QPT processes are encoded as a set of preconditions and effects, creating a rule for achieving a process is straightforward. Similarly, QPT enforces a causal ordering on all quantity changes, permitting compilation of simple operators for influencing quantities.

Sections 4.1 and 4.2 describe the primary expressions of QPT and their compilation into TPLAN rules. Section 4.3 describes rules and operators applicable to all QPT domain models.

4.1 Compilation of Entity Definitions

A QPT entity definition is an expression of the form (DEFENTITY <pattern> . <consequents>) where <consequents> are asserted in every situation in which <pattern> is true. Thus, if we have the following definition

```
''OC'' marks rules and operators constructed by the Operator Compiler.
''USER'' marks rules and operators constructed by the user.
New goals are indented under the solution which introduces them.

GOAL: Increase the amount of water in the liquid state in POT1.
SOLUTION: Apply OC rule that increases POT1's amount of
water by creating a liquid flow from some source and fluid
path. Over the course of planning, ''some source'' is unified
with FAUCET.  The rest of the trace assumes this.
  GOAL: Make POT1 a container.
  SOLUTION: Unify this goal with an initial given.
  GOAL: Make FAUCET a container.
  SOLUTION: Apply OC rule which says liquid sources are
  containers.
    GOAL: Make FAUCET a liquid source.
    SOLUTION: Unify this goal with an initial given.
  GOAL: Get some water into container FAUCET.
  SOLUTION: Unify this goal with an initial given.
  GOAL: Find a fluid path from FAUCET to POT1.
  SOLUTION: Apply an OC rule which says exterior fluid paths
    are fluid paths.
    GOAL: Find an exterior fluid path from FAUCET to POT1.
    SOLUTION: Apply a USER rule which says that an exterior
    fluid path exists when a container is under FAUCET.
      GOAL: Make POT1's location be underneath FAUCET.
      SOLUTION: Apply USER operator to move POT1 from the
      counter to underneath FAUCET.
  GOAL: Make FAUCET's pressure be greater than POT1's pressure.
  SOLUTION: Unify this goal with an initial given.
  GOAL: Make the fluid path's valve position be open.
  SOLUTION: Apply USER operator for opening FAUCET's valve.
```

Figure 4: Trace of a Plan for Adding Water to POT1

```
(DEFENTITY (CONTAINER ?X)
  (HAS-QUANTITY ?X VOLUME)
  (GREATER-THAN (A (VOLUME ?X)) ZERO)
```

and assert (CONTAINER POT1) in certain situations, QPE's inference engine will assert the following in those situations:

```
(HAS-QUANTITY POT1 VOLUME)
(GREATER-THAN (A (VOLUME POT1)) ZERO)
```

Expressing a DEFENTITY as a TPLAN rule is simple. The antecedent and consequents are left unchanged, except that the consequents hold over the same time interval as the antecedent. In the above example, (HAS-QUANTITY POT1 VOLUME) and (GREATER-THAN (A (VOLUME POT1)) ZERO) would hold over the interval over which (CONTAINER POT1) holds. For TPLAN the above DEFENTITY would be expressed as:

```
RULE
  Antecedents: (CONTAINER ?X) $C
  Consequents:
    (HAS-QUANTITY ?X VOLUME) $C
    (GREATER-THAN (A (VOLUME ?X)) ZERO) $C
```

Since TPLAN treats rules as backward chaining operators as well as forward chaining inference rules, TPLAN could use the above rule to achieve goals which unify with any of the consequents.

This scheme is used on all consequents except for qualitative proportionality assertions. For instance, the following definition

```
(DEFENTITY (CONTAINED-LIQUID ?X)
  (HAS-QUANTITY ?X LEVEL)
  (QPROP (LEVEL ?X) (AMOUNT-OF ?X)))
```

gives contained liquids a quantity LEVEL which increases when the AMOUNT-OF contained liquid increases and decreases when

AMOUNT-OF decreases. Encoding the QPROP as a rule consequent would not tell the planner that it can increase a contained liquid's LEVEL by increasing the AMOUNT-OF contained liquid (and likewise for decreasing the LEVEL). Thus, the above DEFENTITY is instead compiled into the following rules.

```
RULE
  Antecedents: (CONTAINED-LIQUID ?X) $cl
  Consequents: (HAS-QUANTITY ?X LEVEL) $cl

RULE
  Antecedents:
   (CONTAINED-LIQUID ?X) $cl
   (INCREASING (AMOUNT-OF ?X) ?CAUSE) $inc
  Temporal Conditions:
    Exists (INTERSECTION $cl $inc) called $int
  Consequents: (INCREASING (LEVEL ?X) ?CAUSE) $int

RULE
  Antecedents:
    (CONTAINED-LIQUID ?X) $cl
    (DECREASING (AMOUNT-OF ?X) ?CAUSE) $dec
  Temporal Conditions:
    Exists (INTERSECTION $cl $dec) called $int
  Consequents: (DECREASING (LEVEL ?X) ?CAUSE) $int
```

The first rule is the result of extracting the QPROP from the DEFENTITY, while the second and third rules encode the extracted QPROP. The second rule says that if a contained liquid exists during $CL, its AMOUNT-OF is increasing over $INC, and $CL and $INC intersect, then the level is increasing over the intersection The third rule is interpreted similarly.

QPE supports a variety of qualitative proportionality expressions (QPROP-, Q=, and Q=) which are handled in similar fashion. The method employed in handling these expressions in DEFENTITY is also used to handle their occurrence in process definitions, which are covered in the next section.

4.2 Compilation of Process Definitions

The antecedents of a process definition (DEFPROCESS) are its individuals, preconditions, and quantity conditions, while the consequents are its process form (such as the PROCESS field of Figure 1), relations, and influences. A DEFPROCESS specifies that in every situation in which facts matching the antecedents hold, the consequents are asserted. This is expressed in TPLAN as a rule which asserts the process form over the temporal intersection of the antecedents. For instance, in Figure 5 the liquid flow process holds over $INT, the temporal intersection of the antecedent intervals ($c1, $c2, $fp, $cl, $vo, and $gt).

The temporal constraints on relations and influences depend on whether they refer to any local quantities of the process. While local quantities hold only during the process, other quantities may exist before, during, and after the process. Thus, relations and influences involving local quantities are asserted over the intersection of antecedents, while all others are asserted over unique intervals containing the intersection. Each of the effects in Figure 5 refer to local quantity FLOW-RATE; thus, they hold over $INT.

As with DEFENTITY, we extract all qualitative proportionality assertions from the DEFPROCESS and create rules which express each of them. Thus, the Q= assertion of Figure 1 would be compiled into separate rules.

```
RULE Process-Liquid-Flow
Antecedents:
  (CONTAINER ?src-can) $c1
  (CONTAINER ?dst-can) $c2
  (FLUID-PATH (FP ?src-can ?dst-can)) $fp
  (CONTAINED-LIQUID (CL ?liq ?src-can)) $cl
  (VALVE-OPEN (FP ?src-can ?dst-can)) $vo
  (GREATER-THAN
    (A (PRESSURE-DIFFERENCE (FP ?src-can ?dst-can))) ZERO) $gt
Temporal Conditions:
  Exists (INTERSECTION $c1 $c2 $fp $cl $vo $gt) called $int
Consequent:
  (LIQUID-FLOW ?src-can ?dst-can ?liq) $int
  (QUANTITY (LOCAL-QUANTITY FLOW-RATE
              (LIQUID-FLOW ?src-can ?dst-can ?liq))) $int
  (INCREASING (AMOUNT-OF (CL ?liq ?dst-can))
              (A (LOCAL-QUANTITY FLOW-RATE
                   (LIQUID-FLOW ?src-can ?dst-can ?liq)))) $int
  (DECREASING (AMOUNT-OF (CL ?liq ?src-can))
              (A (LOCAL-QUANTITY FLOW-RATE
                   (LIQUID-FLOW ?src-can ?dst-can ?liq)))) $int
```

Figure 5: Operator Compiled From LIQUID-FLOW Process Definition of Figure 1

4.3 Universal QPE Rules

QPE encodes a set of universal rules applicable to all physical domains. For instance, value X can not be both equal to Y and greater than Y at the same time. This can be expressed as the following TPLAN rule:

```
RULE
  Antecedents: (equal-to ?x ?y) $=
               (greater-than ?x ?y) $>
  Consequents: $= (:< :> :M :MI) $>
```

The Operator Compiler enforces consistency during planning by including such rules in its compilation of QPT domains. Also included are a set of operators for planning changes in quantities. Section 4.1 described the encoding of qualitative proportionalities, in which rules of the form "If X is increasing then Y is increasing" are output for each qualitative proportionality. These rules allow us to construct operators which achieve inequalities between two quantities by achieving an increase or decrease in one of them. For instance, if we are given that quantity X is greater than quantity Y, we can solve the goal of making X equal to Y by either decreasing X or increasing Y. The latter is encoded as follows:

```
OPERATOR
  Preconditions: (greater-than ?x ?y) $>
                 (increasing ?y ?cause) $increasing
  Consequents: (equal-to ?x ?y) $=
  Constraints: $> (:M) $=
               $increasing (:M :FI :DI :O) $=
```

As an example of planning changes in quantities, suppose that we are given a faucet and stoppered sink containing water. Our goal is to make the water level be greater than SOME-LEVEL. Assume that our domain includes a liquid flow process and a qualitative proportionality stating that a contained liquid's water level increases when the amount of water increases. Figure 6 traces the plan which solves our goal.

5 Discussion

We have described an Operator Compiler which solves simple planning problems in physical domains. The compiler handle

```
GOAL: (GREATER-THAN (LEVEL SINK-WATER) SOME-LEVEL)
SOLUTION: Apply an inequality change operator which
achieves (GREATER-THAN ?x ?y) by increasing ?x.
 GOAL: (INCREASE (LEVEL SINK-WATER) ?cause)
 SOLUTION: Apply QPROP rule which says (LEVEL ?x)
 increases when (AMOUNT-OF ?x) increases and ?x is
 a contained liquid.
   GOAL: (CONTAINED-LIQUID sink-water)
   SOLUTION: Unify with initial given.
   GOAL: (INCREASE (AMOUNT-OF SINK-WATER))
   SOLUTION: Apply liquid flow operator with
   destination SINK and source FAUCET.
```

Figure 6: Trace of a Plan for Changing a Quantity

most syntactic features of QPT. It has been used to compile rules from QPT models of liquid flow, heat flow, and boiling which have allowed generation of simple plans (such as that shown in Figure 1) involving processes, individuals, and changes in quantities. This section describes its current limitations.

The Operator Compiler's primary shortcoming is its overly optimistic strategy for solving goals involving changes in quantities. The strategy makes several naive assumptions:

1. Any positive influence on a quantity causes it to increase, while any negative influence on a quantity causes it to decrease.

2. Any quantity inequality can be achieved by increasing or decreasing one of the two quantities.

Under these assumptions, if a liquid flow into a sink positively influences the water level and a liquid drain negatively influences the water level, assumption #1 fools the planner into thinking the water level is both rising and lowering. Furthermore, assumption #2 fools the planner into thinking the water level will become greater than the sink's maximum level and also lower than the minimum level, assuming that these inequalities are introduced as goals during planning.

These assumptions are made by the operators described in section 4.3 for achieving inequalities. Although they are not valid, they do solve simple problems involving changes in quantities. For instance, given some water draining out of a sink and a goal of filling the sink, the planner can use the inequality operators to generate a plan which turns on the faucet. The solution is partial since nothing specifies that the flow rate must be greater than the drain rate. While such rate ambiguities could be resolved through QPE simulation, many problems involving rates and quantity changes are too complex for our compiled rules, since a rule's effects on a quantity depend on the context. Unfortunately, formulating context-dependent rules and operators is beyond TPLAN's capabilities.

One limitation of the Operator Compiler arises from its dependence on QPT. The compiler is bound by the limits to which QPT can be used to model the physical world. For instance, QPT currently does not provide a mechanism for modeling sets of interacting individuals. (Process definitions explicitly state which individuals affect the process.) Such improvements to QPT would require additional complexity in the compiler and planner.

6 Acknowledgements

Many thanks go to Ken Forbus for providing direction and useful suggestions on this research. Ken Forbus and Brian Falkenhainer gave helpful comments on this document. Discussions with Ken Forbus, Brian Falkenhainer, Barry Smith, and John Collins were helpful in constructing the QPT domain models used in the examples. The Office of Naval Research supported this project through Contract No. N00014-85-K-0225.

References

[Allen, 83] Allen, J.F., "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, vol. 26, pp. 832-843.

[Allen and Koomen, 83] Allen, J.F. and Koomen, J.A., "Planning Using a Temporal World Model", *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 741-747.

[Forbus, 81] Forbus, K.D., "Qualitative Reasoning about Physical Processes", *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, August, 1981.

[Forbus, 84] Forbus, K.D., "Qualitative Process Theory", *Artificial Intelligence* **24**, 1984.

[Forbus, 86] Forbus, K.D., "The Qualitative Process Engine", Technical Report UIUCDCS-R-86-1288, University of Illinois, Department of Computer Science, December, 1986.