From: AAAI Technical Report SS-93-03. Compilation copyright © 1993, AAAI (www.aaai.org). All rights reserved.

Temporal Planning with Constraints (Preliminary Report)

J. Scott Penberthy
IBM T.J. Watson Research Center
P.O Box 704
Yorktown Heights, NY 10598
jsp@watson.ibm.com

Daniel S. Weld*
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98105
weld@cs.washington.edu

Abstract

We present ZENO, an implemented, partial-order planner that handles simultaneous actions occurring over extended time intervals. The action language supports a large subset of KRSL, including metric constraints, deadline goals, and synergistic (additive) effects. We believe the algorithm is both sound and complete. The chief technical contributions are (1) *token reduction* breaks complex goals into pieces that can be systematically supported, (2) incremental algorithms from operations research determine the consistency of temporal and other metric constraints, and (3) lazy evaluation and the Mean Value Theorem are exploited to cope with actions involving continuous change.

1 Introduction

We have built a partial order planner, ZENO, that efficiently handles domain axioms and actions with complex, conditional and metric effects that take place over extended intervals of time. ZENO reasons about simultaneous actions including continuous change, additive effects, external events, and actions that interfere with each other. While other planners exist with combinations of these features, ZENO is different because it is arguably sound and complete, yet reasonably efficient.

The rest of this paper describes our action and plan representation with an example, sketches our planning algorithm, then discusses formal and empirical aspects.

*This research was funded in part by National Science Foundation Grant IRI-8957302, Office of Naval Research Grant 90-J-1904, and a grant from the Xerox corporation. We both thank Tony Barrett, Denise Draper, Oren Etzioni, Keith Golden, Nick Kushmerick, and Mike Williamson for helpful comments.

2 Representation

As with our UCPOP planner [17], ZENO supports both universal and existential quantification in action schemata. Conditional effects and disjunctive preconditions are permitted. A point-based temporal model is used: the state of the world is modeled by a set of *fluents*, *i.e.*, functions from time points to values. Interval constraints are specified with universal quantification over a special `time` variable class.

```
(define (operator fly)
   :parameters (p from to speed)
   :at-time (Ts Te)
   :precondition
      (and (eq (value-at Ts (loc p)) from)
           (≤ 0 speed (max-speed p))
           (forall ((time t))
               (> (value-at t (gas-vol p)) 0))
           (> (value-at Te (gas-vol p)) 0))
   :effects
      ((cause (= speed (/ (distance from to)
                          (- Te Ts))))
      (cause (eq (value-at Te (loc p)) to))
      (forall ((object o))
       when (holds-at Ts (in o p))
       cause (eq (value-at Te (loc o)) to))
      (cause (influence (gas-vol p)
                        (- (/ speed (mpg p)))))))
```

If
The plane p is located at *from* at time Ts;
The *speed* is between 0 and the maximum;
The plane never runs out of gas;
Then
The time of travel is a function of speed and distance;
The plane will be moved to location *from*;
All objects o within the plane will move to *from*;
The gas level will continuously decrease.

Figure 1: This action schema uses universal quantification, conditional effects, metric functions, time, continuous change, synergy, and linear inequalities. The KRSL code at top is described in English below.

A ZENO *action schema* (*e.g.*, figure 1) characterizes a set of possible actions. Note the use of `holds-at` to handle propositional fluents and `value-at` to handle functional fluents. Quantification over time handles interval constraints; the action interval bounds the scope of time t. The categories in a schema are taken from KRSL.

The last effect of `fly` is unique. ZENO `influences` are the quantitative analog to the corresponding concept in Forbus's Qualitative Process theory [6,3]; they allow specification of additive effects through a detailed model of physics. Whenever two or more operators O_1 and O_2 overlap on some time interval, their net effect on the fluent's derivative is calculated by adding the influences.

ZENO *plans* are 3-tuples $\prec\mathcal{S}, \mathcal{C}, \mathcal{L}\succ$ where $\mathcal{S}$ is a set of steps (*i.e.*, instantiated action schemata), $\mathcal{C}$ is a set of constraints and $\mathcal{L}$ is a set of causal links. The steps in $\mathcal{S}$ are partially ordered by the relevant temporal constraints in $\mathcal{C}$.

As in other planners, causal links denote protection ranges, but ZENO's links have two differences: (1) links connect terms (*e.g.*, (F y) and (F x)) rather than whole atomic sentences (*e.g.*, (= (F x) K) and (clear (F y)), (2) ZENO's links capture simultaneity. For example, the link $t_1 \xrightarrow[\{S_i,S_j\}]{(\mathrm{F}\ x)} t_2$ tells us that the only steps constraining the value of (F x) over the interval $[t_1, t_2]$ are S_i and S_j.

Because preconditions and effects have explicit temporal scope, a planning *problem* can be encoded as a partial plan with a *single* dummy step whose time of "execution" bounds all planned activity. This step's preconditions are the plan goals and the effects represent initial conditions, domain axioms, and external events. Since ZENO handles arbitrary linear inequalities and conditional effects, many interesting problems may be expressed. Figure 2 introduces an example that we continue throughout the paper.

```
:at-time (Tstart Tend)
:precondition
   (and (eq JFK (value-at Tend (loc plane3)))
        (≤ (- Tend Tstart) 360))
:effects
   (and (= (value-at Tstart (gas-vol plane3)) 7000)
        (= (mpg plane3) 2)
        (eq SeaTac (value-at Tstart (loc plane3)))
        (eq closed
            (value-at Tstart (status runway)))
        (eq open (value-at maint (status runway)))
        (= maint (+ Tstart 60)))
```

Figure 2: This dummy step encodes the following problem: A plane, initially near Seattle, has 7000 gallons of fuel. It is required to be at an airport in New York in six hours, subject to the fact that the runway is currently closed for scheduled maintenance (to reopen in one hour.)

3 The Zeno Planner

ZENO is a regressive, causal link planner that builds on our previous work with UCPOP [17], a sound and complete partial order planner which handles universally quantified actions with conditional effects [13]. Like SNLP [11] and UCPOP, ZENO's main loop makes two types of choices:

1. Supporting unachieved preconditions with the effects of new or existing steps and recording this commitment with a causal link, and

2. Resolving threats to existing causal links (*i.e.*, third steps whose effects may cause harmful interactions) via promotion, demotion, *etc.*

In ZENO, item 1 finds an initial effect that constrains some variable in a precondition. Item 2 corresponds to deciding which other effects should simultaneously constrain the same variable. The algorithm first tries to find a set of effects *consistent* with the goals, iterating between items 1 and 2 until all variables are constrained. ZENO then checks whether the effects satisfy the preconditions, *i.e.*, whether the effects are *valid*. If not, those unsatisfied preconditions are treated as new subgoals and planning continues as before. Only when a consistent, valid set of effects is found does ZENO halt.

ZENO uses the techniques developed in UCPOP for handling quantifiers, conditional effects and codesignation constraints. A new method is introduced for matching effects against preconditions. A suite of algorithms from linear programming solve constraints and verify plan consistency. Lazy evaluation handles continuous change.

3.1 Matching Effects to Preconditions

In many planners, open goals are unified with step effects to see if the step can achieve the goal. For example, since the goal (on A B) unifies with the effect (on x y), a planner knows that (puton x y) is a possibly relevant action. Furthermore, in SNLP-style planners, if (ON A B) is supported by an unthreatened link, the planner knows that it will be satisfied. However, if preconditions and effects include arbitrary metric equations this simple scheme does not suffice. For example, given the goal (< A B) it is not immediately apparent that the effect (> C 0) is relevant, but it *is* if other steps have (= A C) and (= B (* 2 C)) as effects. Together, these three effects satisfy the goal even though none of them unify with it.

To plan to achieve metric preconditions, ZENO needs

a fundamentally different technique, *token reduction*, which works as follows:

1. All preconditions and effects are rewritten into simpler constraints by replacing *tokens* with corresponding *token variables*,
2. Tokens in precondition constraints are supported by causal links to matching tokens in the effects of new or existing steps,
3. When causal links are formed, constraints are added to $\mathcal{C}$,
4. When all tokens are supported, and $\mathcal{C}$ is consistent, and the effect constraints in $\mathcal{C}$ entail the precondition constraints in $\mathcal{C}$, then ZENO returns the successful plan.

Functional expressions which denote domain specific constants or fluent dereferences *e.g.*, `(distance ...)` and `(value-at ...)`, are called *tokens*. ZENO replaces all tokens inside precondition and effect constraints with *token variables*, V_n. For example, `fly`'s precondition

`(> (value-at` *Te* `(gas-vol plane3)) 0)`

is rewritten as `(>` V_1 `0)`, a constraint which conveniently contains only variables and constants. This constraint is posted to $\mathcal{C}$.

Next, ZENO supports all tokens in the precondition constraint with a link from a token in a preceding effect. Suppose that ZENO selects an effect of a newly created `fly` step, S_1; we show both its original[1] and token-reduced forms:

```
(= (value-at Te (gas-vol p))
   (- (value-at Ts (gas-vol p))
      (* (- Te Ts) (/ speed (mpg p)))))

(= V2
   (- V3
      (* (- Te Ts) (/ speed V4))))
```

The descriptor of V_2 unifies with that of V_1 provided that `(eq` p `plane3)`. ZENO creates a causal link $Ts \xrightarrow[\{S_1\}]{(\text{gas-vol } p)} Te$, stores the relevant codesignations and temporal constraints in $\mathcal{C}$, and equates the token variables of the effect and precondition (*e.g.*, $V_1 = V_2$ is added). The process repeats as token reduction is applied to the preconditions of step S_1 and other, unbound tokens in the effect. This results in more constraints on V_3 and V_4.

After all unbound tokens have been linked, ZENO asks whether the chosen effects satisfy the initial precondition (via constraint satisfaction in $\mathcal{C}$). If they do, then the goal is satisfied. If not, token reduction is tried again to further constrain the unsatisfied precondition. The current implementation delays these validity tests until all goals have been processed.

Interestingly, token reduction also works for symbolic goals. Since a `holds-at` precondition is just an ordinary token, the resulting links are isomorphic to those of UCPOP. Furthermore, codesignations `eq` and `neq` can be handled in the same way as regular metric constraints.

3.2 Efficient Constraint Management

Since ZENO pushes all temporal and metric aspects into $\mathcal{C}$, sound and efficient constraint handling is absolutely critical to the planning algorithm. ZENO requires two types of computation from the constraint reasoner: (1) test if $\mathcal{C}$ is inconsistent, and (2) test if precondition and goal constraints are necessarily guaranteed by effect constraints. Specialized algorithms cooperate to handle the different types of constraints in $\mathcal{C}$: codesignations, linear equalities, and inequalities. Nonlinear equations are delayed until they become linear.

ZENO tests the consistency of $\mathcal{C}$ to see if it must backtrack, handling different constraints with different mechanisms. Codesignations are handled as they were in SNLP and UCPOP. Mathematical formulae posted by ZENO are parsed (using LALR reduction) into a set of equations, inequalities, and pairwise nonlinear equations. For example, assuming that the `distance` token is renamed to V_3, `fly`'s effect

`(=` *speed* `(/ (distance` *from to*`) (-` *Te Ts*`)))`

is parsed into three equations,

$$\begin{aligned} 0 &= x_2 - speed \\ 0 &= x_1 + Ts - Te \\ x_2 &= \frac{V_3}{x_1} \end{aligned}$$

which include 2 internal variables, x_n. The resulting linear equalities are solved by Gaussian elimination, linear inequalities by the Simplex algorithm, and nonlinear equations are delayed. Determining inconsistency using Gaussian elimination is straightforward, but our use of Simplex is more subtle. Recall that linear programming is the task of minimizing a cost function while satisfying a set of linear inequalities [10]. The Simplex algorithm starts by constructing a *polytope* (*i.e.*, a convex region in $\Re^n$) which exactly covers the set of solutions to the linear inequalities. Next, it walks along vertices of the polytope in search of values that minimize the cost function. While exponential in the worst case *i.e.*, it may have to traverse all vertices, the algorithm is extremely fast in practice. For ZENO, the optimization aspect is irrelevant — we're only interested in whether the polytope is malformed, *i.e.*, if it vanishes to a point, no solutions exist and the constraints are inconsistent. For maximum speed, we use

[1] See 3.4 to see how influences are transformed into constraints similar to these.

Jaffar *et.al.*'s [9] version of Simplex which has been optimized for incremental updates.

3.3 Threat detection

Because ZENO's causal links mark the range in which a token is being protected, there are many ways for other steps to interfere and these must be checked carefully. In particular, once ZENO commits to a causal link, $t_1 \xrightarrow[\{s_i,s_j\}]{(\mathrm{F}\ x)} t_2$, *any* constraint C_i that references (F y) which possibly codesignates (F x) at some time $t_i \in [t_1, t_2]$ might affect the consistency of C. This is a *threat.*

ZENO must decide whether each C_i is to be a simultaneous constraint on (F x), or whether it should be constrained out of the way. This is "threat resolution" in ZENO, where the algorithm must choose to either (1) *promote* by constraining $t_i > t_2$, (2) *demote* by constraining $t_i < t_1$, (3) *separate* by adding a neq constraint so that the token descriptors don't match, (4) if C_i is conditional, *confront* by constraining $t_1 \leq t_i \leq t_2$ while adding the negation of C_i's precondition to the goal agenda, or (5) *embrace* the threat by constraining $t_i = t_1$ and including C_i as a new constraint on (F x). The fifth alternative is the only difference between ZENO and UCPOP; it corresponds to adding a new simultaneous constraint.

3.4 Continuous change

There are two types of continous change in ZENO: interval constraints that combine simultaneously and those that combine additively (influences). The former are specified as universal quantifications over some time t. Default bounds of the interval are calculated from the action interval $[t_1, t_2]$. Semi-closed intervals $[t_1, t_2)$ are assumed for preconditions, open intervals (t_1, t_2) for effects. Consider the precondition of `fly`:

```
(forall (time t)
    (> (value-at t (gas-vol veh)) 0)))
```

Although the value of `gas-vol` may vary continuously through time, ZENO requires all change to be piecewise linear. This assumption, combined with the Mean Value Theorem, guarantees that the extrema occur at the endpoints of each subinterval. By checking constraints at all possible subinterval boundaries, ZENO ensures that they are satisfied over the whole interval.

These "boundaries" are found by examining all causal links, then extracting those timepoints which possibly fall within the constraint interval. For each such timepoint, ZENO either adds a new effect constraint (for effects) or adds a new subgoal (for preconditions). In addition, effects are generated on the fly when trying to satisfy tokens whose time falls within the constraint interval.

Influences specify continuous change in a more modular way: the derivative of a quantity at time t equals the sum of all influences in effect at t [6,5]. ZENO computes these sums on the fly, by choosing to satisfy a token with a *set* of influences, not just one. The combined constraints are then treated as a single interval constraint. Subtleties with this approach are discussed in [16].

Since there is no bound on the number of simultaneous influences that may participate in satisfying a goal, ZENO faces a search space with an infinite branching factor. The implementation explores this space by iterative expansion — it gradually increases the cardinality of the set of steps considered. Singleton sets correspond to the case where synergy is excluded.

4 Related Work

Because of our interest in formal properties, ZENO is closest in spirit to the work of Allen, Chapman, McAllester, and Pednault. Allen and Pelavin [15,2,1] describe an elegant theory of temporal planning based on first order logic and an interval model of time. While their framework is expressive and precise, it does not specify an actual planning algorithm which is liberated from the expense of theorem proving and NP-hard temporal constraint propagation.

Numerous systems with some of ZENO's features have been built in the past twenty years [8,19,18,21,4] and we have drawn insight from many of them. Simmons' GORDIUS [18] handled actions with conditional and metric effects, but was incomplete and had a state-based model of time. Our approach is considerably simpler than that of SIPE [21] — ZENO avoids SIPE's parallel links and its complex traversal schemes. Similarly, we believe that ZENO's treatment of simultaneous and synergistic effects is more general than SIPE's. While OPLAN uses ideas from operations research to optimize resource usage, they use different techniques for temporal management. In contrast, ZENO uses an integrated approach for both temporal and other metric constraints, but makes no claim to handle resources efficiently.

We have drawn both ideas (using Gaussian elimination and Simplex Phase I iteration to manage linear equalities and inequalities) and algorithms (incremental versions of these combinatorial algorithms) from the CLP($\Re$) [7] implementation of Jaffar *et. al.* [9]

5 Conclusion

We have described the ZENO planner which handles a rich, temporal language with simultaneous and synergistic actions. The problem of matching constraints between effects and preconditions was solved using a new technique called *token reduction.* The Simplex algorithm, Gaussian elimination, and nonlinear delaying techniques efficiently determine the consistency of constraints. Assuming that all constraints are piecewise

linear, ZENO reduces continuous change to constraints on timepoints through lazy evaluation.

Although we have not completed a formal analysis of ZENO, we believe that ZENO can be shown both sound and complete. Precise definition of these terms requires some extension to Pednault's definitions [12,14] to account for metric time and floating point precision. We believe that ZENO is sound as long as all nonlinear constraints have been linearized, because all its underlying constraint handlers are sound and complete. Furthermore, we believe that ZENO can be shown complete using an extension of UCPOP's completeness proof [17]; however, since many details remain to be worked out, these claims should be taken as a statement of intent guiding our design of ZENO.

ZENO's Common Lisp implementation is almost complete[2] — it has solved numerous problems including the `fly` example of figure 2 (1.6 seconds) and the classic "Two Jugs" problem (95 seconds). Efficiency is achieved by using the constraint handlers on demand. Caches maintain temporal inequalities, *e.g.*, $t_1 < t_2$ and constant metric values, *e.g.*, $V = 3$, for the slew of queries required by ZENO. Early results show that ZENO performs as fast as UCPOP on domains without metric aspects (*e.g.*, approx 10ms per plan refinement). Medium sized problems with constraints (*e.g.*, between 5 and 10 steps with 100 or more constraints) rarely take more than a few minutes to solve. See [16] for a complete description and empirical analysis.

References

[1] J. Allen. Planning as temporal reasoning. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 3–14, 1991.

[2] J. Allen, H. Kautz, R. Pelavin, and J. Tenenberg. *Reasoning about Plans.* Morgan Kaufmann, San Mateo, CA, 1991.

[3] J. Crawford, A. Farquhar, and B. Kuipers. QPC: A Compiler from Physical Models into Qualitative Differential Equations. In *Proceedings of AAAI-90*, pages 365–372, August 1990.

[4] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, November 1991.

[5] E. Davis. Axiomatizing Qualitative Process Theory. In *Proceedings of KR-92*, October 1992.

[6] K. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24, December 1984. Reprinted in [20].

[7] Nevin Heintze, Joxan Jaffar, Spiro Michaylov, Peter Stuckey, and Roland Yap. The CLP($\mathcal{R}$) Programmer's Manual Version 1.1. Technical report, IBM T.J. Watson Research Center, November 1991.

[8] G. Hendrix. Modeling Simultaneous Actions an Continuous Processes. *Artificial Intelligence*, 4:145–180, 1973. Reprinted in [20].

[9] Joxan Jaffar, Spiro Michaylov, Peter Stuckey, and Roland Yap. The CLP($\mathcal{R}$) Language and System. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.

[10] H. Karloff. *Linear Programming.* Birkhäuser, Boston, 1991.

[11] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proceedings of AAAI-91*, pages 634–639, July 1991.

[12] E.P.D. Pednault. *Toward a mathematical theory of plan synthesis.* PhD thesis, Stanford University, December 1986.

[13] E.P.D. Pednault. ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In *Proceedings Knowledge Representation Conf.,*, 1989.

[14] E.P.D. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *Proceedings IJCAI-91*, July 1991.

[15] R. Pelavin and J. Allen. A Model for Concurrent Actions having Temporal Extent. In *Proceedings of AAAI-87*, pages 246–250, 1987.

[16] J.S. Penberthy. *Principled Planning with Expressive Languages.* PhD thesis, University of Washington, 1993.

[17] J.S. Penberthy and D. Weld. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proceedings of KR-92*, pages 103–114, October 1992.

[18] R. Simmons. Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems. AI-TR-1048, MIT AI Lab, September 1988.

[19] S. Vere. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 5, 1983.

[20] D. Weld and J. de Kleer, editors. *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann, San Mateo, CA, August 1989.

[21] D. Wilkins. Can AI planners solve practical problems? *Computational Intelligence*, 6(4):232–246, November 1990.

[2] Although the test for constraint redundancy and composition of multiple, simultaneous influences are only partially implemented, all other features are complete.