Implementing a qualitative reasoner: Part 2

EECS 344
Winter 2008
Why Qualitative Physics?

• Suppose someone tells you that the level in G is rising, and you want to figure out what could be happening.
Qualitative Process Theory

- Ontological Assumptions
- Mathematics
- Causal Account
- Organizing Domain Theories
- Basic Inferences
Example

Three possible contained stuffs, four potential fluid flows
Example
Design issues

• How should we represent changes over time?
• What should the modeling language look like?
• How do we build scenario models?
• How should inequality reasoning be performed?
• How should we search for interpretations?
Representing change over time

• In this task, we don’t need to!
• Several good alternatives if we did:
  – Modal operators \(\text{Holds } p \ t\)
  – Slices \(>(P\ \text{at } Wg\ t1))\ (P\ \text{at } Wg\ t2))\)
  – Implicit temporal notation
    \(>(P\ Wg)\ (P\ Wf))\)
The Modeling Language

- `defprocess`, `defview` to define entities and relationships that change over time
- Implement similarly to integration operators in JSAINT
- Need three other constructs as well
defPredicate

• Provides easy way to define the consequences of a predicate

• (defPredicate <form> .
   
   <consequences>)

( defePredicate (heat-connection ?src ?path ?dst)
  (heat-path ?path) ;; inferred type
  (heat-connection ?dst ?path ?src)) ;; symmetric
defEntity

- Provides a way of defining new entities
- Implication: Predication true if and only if the entity exists.

(defEntity (<predicate> <ind>) . <consequences>)

(defentity (Physob ?phob)
    (quantity (heat ?phob))
    (quantity (temperature ?phob))
    (> (A (heat ?phob)) ZERO)
    (> (A (temperature ?phob)) ZERO)
    (qprop (temperature ?phob) (heat ?phob)))
defRule

• Provides “glue” for other descriptions
  • (defrule <name> <triggers> . <consequences>)
  • (defrule Contained-Stuff-Existence
      ((Container ?can)(Phase ?st)(Substance ?sub))
      ;; Assume that every kind of substance
      ;; can exist in every phase inside
      ;; every container.
      (quantity ((amount-of ?sub ?st) ?can))
      (>= (A ((amount-of ?sub ?st) ?can)) ZERO))
Subtle issue: Existence of quantities

- Continuous properties of things that don’t exist need to be treated differently.
  - The rat poison in your coffee.
  - The radiation level of the plutonium under your chair

- How do we easily link quantities to individuals?
Linking quantities to individuals

- Declare them explicitly
  `(defquantity-type (heat individual))`

- Force them to be unary
  `(heat <fluid>)`

- Can *curry* to allow multiple arguments
  `((amount-of-in <substance> <phase>) <container>)`
Building Scenario Models

- Domain Theory
- Structural Description
- Task Constraints
- Scenario Model

Model Builder
Working Assumptions

• All of the situation is relevant
  – No subsystems that can be ignored or isolated.
  – Can ignore my car’s electrical system when trying to fix a leak in the radiator.

• All of the domain theory is relevant
  – No phenomena that can be ruled out a priori.
  – Quantum tunneling as an explanation for why my car is using gas unusually quickly

• The domain theory will introduce only a finite number of individuals, given a finite structural description
  – Every physical object can be broken down into at least two parts, each of which itself is a physical object.
Solution: Instantiate everything

- Translate domain theory into LTRE rules.
- Enter structural description as assumptions (or assertions)
- Let LTRE sort it out.
The logic of processes

• Let’s take a look at the code...
  – \texttt{mlang.lsp} implements the constructs of the modeling language
  – \texttt{tnst.lsp} implements a sample domain theory.
Efficient inequality reasoning

• How not to do it:
  (rule ((:true (> ?n1 ?n2) :var ?>1))
  (rule ((:true (> ?n2 ?n3) :var ?>2))
    (rassert! (:implies (:and ?>1 ?>2)
      (> ?n1 ?n3))
    :transitivity))
  (rule ((:true (= ?n2 ?n3) :var ?=1))
    (rassert! (:implies (:and ?>1 ?=1)
      (> ?n1 ?n3))
    :transitivity))
  ;; Plus two similar rules

• Introduces new, unnecessary intermediate statements
What’s really needed?

- Key observation: Only inequalities mentioned by some other part of the scenario model are relevant.
- Treat inequalities as a graph.
- All transitivity inferences correspond to cycles in the graph.

![Diagram](attachment:image.png)

Implied by other two relationships
Further Optimization: "Soft inequalities"

- Obvious representation takes four statements
  
  \[ A < B \]
  \[ A = B \]
  \[ A > B \]
  \[ A \perp B \]

- Lots of redundancy
How soft inequalities work

• Really only need two statements per comparison:

\[ A < B \iff A \leq B \land \neg B \leq A \]
\[ A = B \iff A \leq B \land B \leq A \]
\[ A > B \iff \neg A \leq B \land B \leq A \]
\[ A \perp B \iff \neg A \leq B \land \neg B \leq B \]
Let’s look at the inequality code

- \texttt{ineqs.lsp} defines the transitivity code
Searching for interpretations

• What’s an interpretation?
  – A set of active processes and their combined effects that predicts the observed data.

• A form of abductive inference
  – “If these processes were acting, and this change went this way instead of that, then we’d get what we are seeing.”
  – Given B, A implies B, infer A.

• Constraint: Want the most plausible interpretation.
  – The level is rising because gravity within the container just changed its sign
How to search process structures?

• Use dependency-directed search
• But over what?
  – set of preconditions and quantity conditions?
  – set of active processes and views?
• Many combinations of preconditions and quantity conditions have equivalent process structures
• Simpler to organize search around set of active views and processes.
How the search is organized

• Driver routine that organizes everything else
  – mi.lisp
• Generation of all process structures and view structures
  – psvs.lisp
• Resolve influences for each
  – resolve.lisp
• Recording complete states
  – states.lisp
Let’s look at the search code...
Resolving Influences

• Find construals for the sets of influences on all quantities
  – SETUP–IR

• Impose a causal ordering on all the quantities
  – FIND–INFLUENCE–ORDERING

• Starting with direct influences, attempt to resolve all quantities.
  – RESOLVE–INFLUENCES–ON

• Use dependency-directed search to find consistent choices when ambiguity arises
  – RESOLVE–COMPLETELY
We won’t look at the influence resolution code

• You’ll do that as part of your homework
Implementing QP Laws

• Use PDIS rules to implement simple universal laws
• Use PDIS rules to provide “glue” linking lisp procedures to the rest of the system.
• Let’s examine laws.lisp...
Some design observations

• Sophisticated non-monotonic reasoning is quite feasible
  – *qualification problem* (what can affect a situation) solved by theory of what kinds of mechanisms can be causes.
  – *frame problem* solved by presuming that things only change when caused.
  – Logicians running behind practice, as usual
Tradeoff: What’s in rules versus procedures?

• Some decisions cannot be made locally
  – Closed world assumptions

• Need flexible control structures that can make global decisions
  – Surely there is something better than Lisp code for this!
Migration of rules to special-purpose code

• Examples
  – Reasoning about ordinal relations
  – Influence resolution

• Do “obvious” implementation first

• Optimize only when you know where the bottlenecks are
Habitability

- Make formats for knowledge as implementation-independent as possible
- Make readable output and reports early
- When the going gets tough, the tough get GUI
Homework 6

- Assigned 2/14/08
- Due by start of class 2/21/08
- Please use subject line HW6
- From *Building Problem Solvers*, Chapter 11:
  - Problem 3
  - Problem 13
  - *Extra credit*: Problem 10