

---

---

# PowerLoom Overview, Features and Examples

Hans Chalupsky  
Project Leader, USC/ISI Loom KR&R Group

# Overview

---

---

- Logic-based KR&R
  - What is it, advantages, disadvantages
- PowerLoom
  - Quick overview
  - Basic features
  - Tutorial
  - Advanced features relevant for scientific reasoning
  - Debugging failed inferences
- Conclusion

---

---

# Logic-Based Knowledge Representation & Reasoning

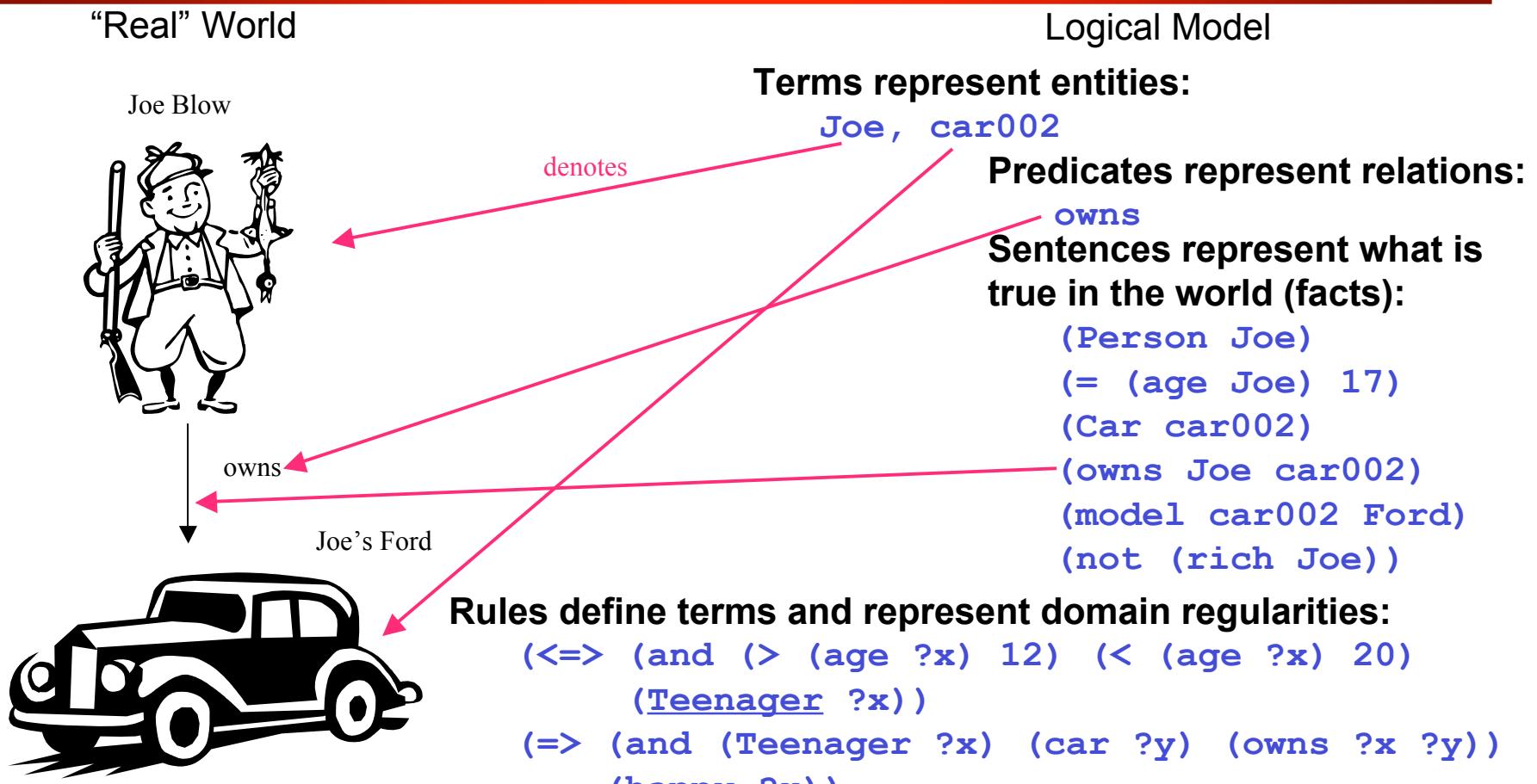
# Logic-Based KR&R

---

---

- Knowledge Representation & Reasoning
  - Studies a wide variety of paradigms & algorithms for
    - Modeling salient aspects of a world of interest
    - Reasoning with such models
- There are many different modeling paradigms
  - Logic-based, frame-based, graph-based, probabilistic, etc.
- PowerLoom is a logic-based KR&R system
- How do logic-based models represent the world?

# Logical Models 101



**Facts + rules + inference derive concluded facts:**

(Teenager Joe)  
 (happy Joe)

# How Does Logic Model the World?

---

- Terms correspond to entities in the (some) world
- Predicates model properties and relations between entities
- Domain rules define and constrain relations, for example, “If Joe is a teenager who owns a car then Joe is happy”
- Logical inference rules define the propagation of truth between logical sentences, for example:  
from  $X$  and  $X \Rightarrow Y$  it must be true that  $Y$
- The more rules and sentences we add, the higher constrained their “interpretation” (what they could mean) becomes
- However, every consistent theory always has infinitely many (formal) interpretations

# Advantages of Logic-based Models

- Tradition
  - Well-understood syntax and semantics
  - Very large amount of relevant research (> 2000 yrs.)
  - Many available logic-based tools
    - Provers, constraint reasoners, learners, planners, KR&R systems, etc.
- Representational power
  - Negation
  - Disjunction
  - Equality (object identity)
  - Logical connectives
  - Quantification
  - Rules, constraints
  - Abstraction
  - Definitions
  - Extendable vocabulary, ontologies
  - If you can't say it in logic, you probably don't want to say it 😊

# Advantages of Logic-based Models

---

- General purpose, well-understood inference mechanisms
  - Deduction
  - Abduction
  - Induction
  - Constraint satisfaction
  - Automated reasoners

# Advantages of Logic-based Models

---

---

- Formalizes reasoning and gives justification
  - Proofs provide justifications for derived facts
  - If one accepts the premises one must/should accept the conclusions
- **Explanation and understandability**
  - Proofs are a good starting point to provide explanations
  - Logical models are “easy” to understand and interpret (e.g., rules learned by an ILP method)
  - Logical models are easier to debug than other approaches
- Translatability
  - Different logical representations are (often) easily translatable into each other (e.g., this diffuses the attribute-vs.-collection distinction)

# Disadvantages?

---

---

- Disadvantages
  - Difficult to handle uncertainty and probabilistic reasoning
    - But, various efforts to combine logical and probabilistic models (e.g., PRM's)
  - Complexity of reasoning algorithms
  - Sometimes too expressive, too many different ways of saying the same thing
  - Hard to handle grey areas, but the world is grey
    - Have to make hard decisions (true, false)
    - Hard to say “many”, “few”, “nearly”, etc. (frustrates NLP people)

---

---

# PowerLoom

# PowerLoom KR&R System

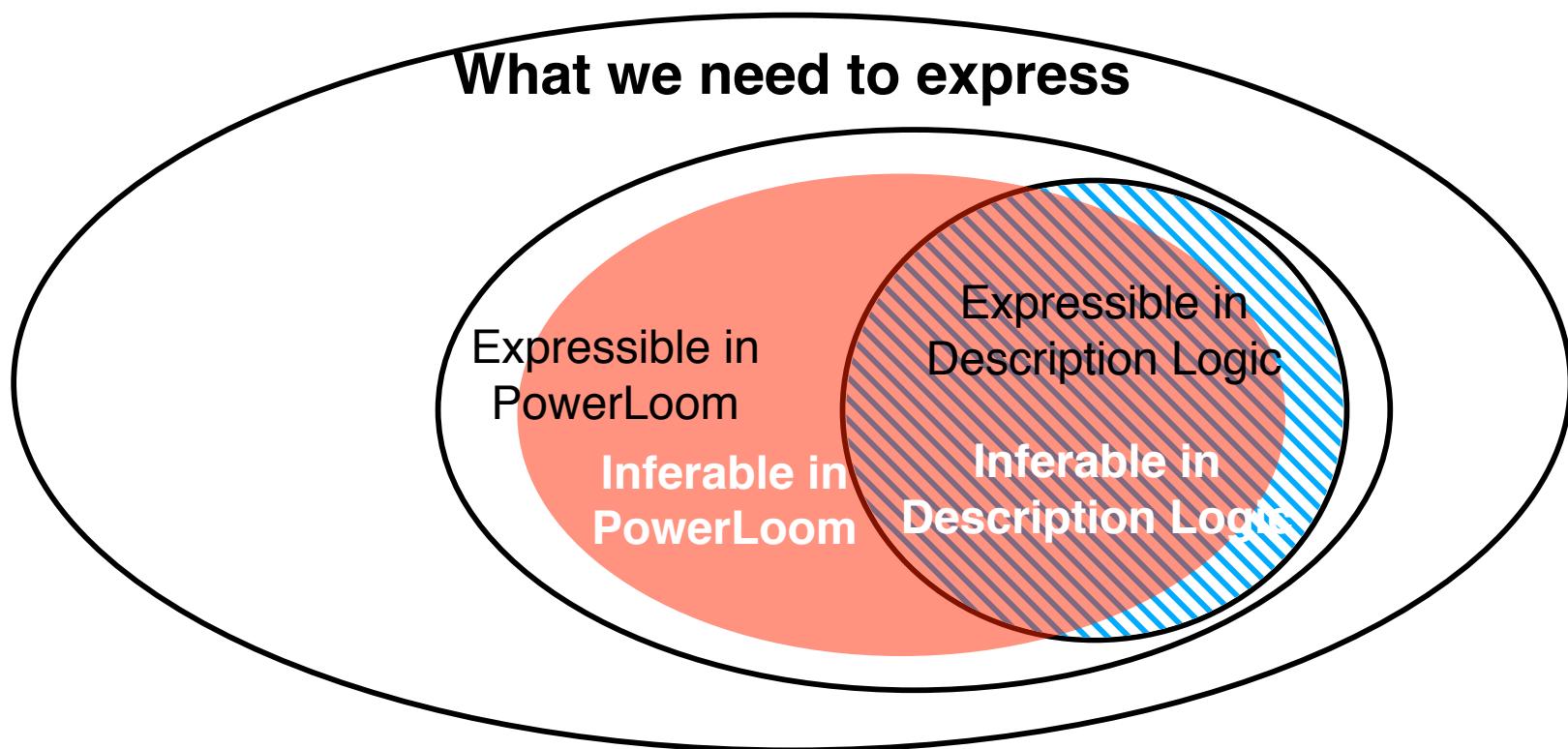
---

---

- Successor to the successful Loom KR&R system with
  - More expressive representation language
  - Less arcane syntax
  - Better scalability
  - Better portability & extensibility (available in Lisp, C++, Java)
- Based on first-order predicate logic
- Not a description logic but has description logic features
  - E.g., classifier, type and cardinality reasoning, subsumption
- Focus on expressivity + scalability
- Pragmatic stance
  - Usability is more important than theoretical “neatness”
  - Expressivity is more important than inferential completeness

# Expressivity vs. Inferential Completeness

- Qualitative comparison of KR-system philosophies
  - Description logics: restricted expressivity, sound, complete, tractable
  - PowerLoom: representationally promiscuous, sound, 80/20 complete & tractable (handle most expected inferences in reasonable time)



# Inference Capabilities

---

---

- Query engine to retrieve asserted and inferable statements
- Prolog-style backward inference enhanced by
  - Recursive subgoal detection
  - Proper handling of negation
  - Hypothetical reasoning
  - Resource-bounded depth-first or iterative deepening search
  - Proof tree (justification) recording
- Forward inference and simple constraint propagation
- Equality and inequality reasoning
- Subsumption reasoning + relation & instance classification
- Partial match for reasoning with incomplete information
- “WhyNot” abductive inference for query diagnosis
- Extensible reasoning specialists architecture

# Knowledge Base Management

---

- Incremental monotonic and non-monotonic updates
  - Interleave definitions, assertions, retractions with retrieval and inference
  - Truth maintenance via inference caches
- Context mechanism
  - Separate name and assertion spaces with inheritance
  - Provides powerful structuring mechanism for KBs
  - Facilitates scenarios and hypothetical reasoning
- Simple load and save KB mechanism via files
  - Experimental RDBMS persistence in the works

# Built-in Ontologies

---

- PL-KERNEL-KB
  - Minimal upper level that defines the core representational vocabulary
    - Relation, concept, function, set, holds, proposition, range-cardinality, ...
  - Emphasis on minimality
    - Represent what's absolutely necessary to make PowerLoom work
    - The less it contains, the less opportunities for modeling conflicts there are when preexisting ontologies get imported
    - Not yet minimal enough, various things still need to be extracted into their own loadable ontology
- Other available ontologies
  - Small time ontology: supports time points, durations, temporal arithmetic
  - Units ontology: units support loadable on demand
  - Various translations of upper models (e.g., Cyc, SENSUS), application ontologies (e.g., EELD, Seismology, ...)

# Tools and APIs

---

---

- Ontosaurus KB browser
  - Web-based, dynamic generation of HTML pages viewable in standard browser
- PowerLoom GUI
  - Java-based browse/edit/query environment
  - Client/server based, deployable via Java WebStart in standard browser
- Interactive command-line interface
- Programmatic PowerLoom Interface (PLI)
  - Lisp, C++ and Java bindings
- Lisp-based Loom API
  - Facilitates import of legacy Loom KBs
- OntoMorph translation system
  - Facilitates import of KBs in other languages (e.g. Flogic)
- Initial Semantic Web support
  - Import translator for RDF/RDFS

# PowerLoom GUI

KB Edit Objects Query View Help

 Concepts for AIRCRAFT-KB

Modules: ROOT-MODULE, SEISMOLOGY, POWERLOOM, PL-KERNEL, CYC-FR, KIF-FRA, LOOM-A, PL-USER, AIRCRAFT-KB

Propositions:

**Propositions for SAN-ANDREAS- CHOLAME**

- P (= (THE-FAULT-SLIP-RATE SAN-ANDREAS-CHOLAME) (THE-FAULT-SLIP-RATE SAN-ANDREAS-CHOLAME))
- P (= (THE-FAULT-MAXIMUM-MAGNITUDE SAN-ANDREAS-CHOLAME) 6.9)
- P (= (THE-FAULT-CHARACTERISTIC-RATE SAN-ANDREAS-CHOLAME) (THE-FAULT-CHARACTERISTIC-RATE SAN-ANDREAS-CHOLAME))
- P (= (THE-FAULT-RAKE SAN-ANDREAS-CHOLAME) (THE-FAULT-RAKE SAN-ANDREAS-CHOLAME))
- P (= (THE-FAULT-DIP SAN-ANDREAS-CHOLAME) (THE-FAULT-DIP SAN-ANDREAS-CHOLAME))



Query: (FAULT ?fault)

Module: SEISMOLOGY

Results:

- ?fault
- NORTHRIDGE-FAULT
- SAN-ANDREAS-1857-RUPTURE
- SAN-ANDREAS-CARRIZO
- SAN-ANDREAS-CHOLAME
- SAN-ANDREAS-COACHELLA
- SAN-ANDREAS-MOJAVE
- SAN-ANDREAS-PARKFIELD-SEGMENT
- SAN-ANDREAS-SAN-BERNARDINO
- SAN-ANDREAS-SOUTHERN

Execute Cancel

 Instances for AIRCRAFT-KB

RELATION-ROOT: 20MM-VILLCAN

 Instances for SEISMOLOGY

RELATION-ROOT: QUARTERNARY

ALTITUDE-OF, DEPTH-OF, DERIVATIVE-FN, LATITUDE-OF, LONGITUDE-OF, MEASURED-ATTRIBUTE, MEASURED-LOCATION, MEASUREMENT-DEPTH, MEASUREMENT-UNCERTAINTY, NEHRP-CLASS-LETTER, PARAMETER-CHARACTERISTIC, PARAMETER-DESCRIBES, PARAMETER-OBJECT-TYPE, PARAMETER-VALUE, QUANTITY-AMOUNT, QUANTITY-OF

Instances for SEISMOLOGY:

- I NORTHRIDGE-EARTHQUAKE
- I NORTHRIDGE-FAULT
- I NORTHRIDGE-HYPOCENTER
- I QUARTERNARY
- I SAN-ANDREAS-1857-RUPTURE
- I SAN-ANDREAS-CARRIZO
- I SAN-ANDREAS-CHOLAME
- I SAN-ANDREAS-COACHELLA
- I SAN-ANDREAS-FAULT-SYSTEM
- I SAN-ANDREAS-MOJAVE
- I SAN-ANDREAS-PARKFIELD-SEGMENT
- I SAN-ANDREAS-SAN-BERNARDINO
- I SAN-ANDREAS-SAN-BERNARDINO
- I SAN-ANDREAS-SOUTHERN



Name: WEIRD-FAULT

SuperConcepts: FAULT

Documentation: The class of very strange faults.

Module: SEISMOLOGY

Relations:

Propositions:

Rules:



Search String: north

Module: SEISMOLOGY

Contains, Starts With, Ends With, Case Sensitive, Exactly Matches

Objects:

Object	Module	Type
NORTHRIDGE-EARTHQUAKE	SEISMOLOGY	INSTANCE
NORTHRIDGE-FAULT	SEISMOLOGY	INSTANCE
NORTHRIDGE-HYPOCENTER	SEISMOLOGY	INSTANCE

OK Cancel

Current Module: AIRCRAFT-KB

# Ontosaurus Browser

**Module:** SEISMOLOGY    **Show**    **Hold Window**

**any**    basin    **Find**    **Match**    Substring

**Relation BASIN-DEPTH-2.5**

**Textual Definition**

Module: [SEISMOLOGY](#)  
 Seen from: [SEISMOLOGY](#)  
 Depth to the 2.5km/s Vs boundary in a basin.

**Types**

[DEPTH-MEASURE](#)    [DISTANCE-MEASURE](#)    [QUANTITY](#)  
[FUNCTION](#)    [RELATION](#)    [SET](#)  
[DUPLICATE-FREE-COLLECTION](#)    [COLLECTION](#)    [AGGREGATE](#)  
[DUPLICATE-FREE](#)  
[SITE-PARAMETER](#)    [PARAMETER](#)

**Subrelations**  
 None

**Superrelations**  
 None

**Structured Description**

BASIN-DEPTH-2.5

**Facts**

(DOCUMENTATION BASIN-DEPTH-2.5 "Depth to the 2.5km/s Vs boundary in a basin.")  
(DUPLICATE-FREE BASIN-DEPTH-2.5)  
(NTH-DOMAIN  
 BASIN-DEPTH-2.5  
 0  
 SITE)  
(NTH-DOMAIN  
 BASIN-DEPTH-2.5  
 1  
 THING)  
(= (PARAMETER-DESCRIBES BASIN-DEPTH-2.5) sk1246//SITE)  
(PRETTY-NAME BASIN-DEPTH-2.5 "Basin\_Depth")

(BASIN-DEPTH-2.5)  $\Leftrightarrow$  (BASIN-DEPTH-2.5)

**Rules**

(FORALL (?S)  
 ( $\Rightarrow$  (EXISTS (?V57)  
 (AND  
 (= (BASIN-DEPTH-2.5 ?S) ?V57)  
 (= (UNITS 0 "m") ?V57))))  
 (EXISTS (?V58 ?V59)  
 (AND  
 (= (VS30-OF-SOIL ?S) ?V58)  
 (= (UNITS 2.5 "km/s") ?V59)  
 (UNIT? ?V58 ?V59))))

BASIN-DEPTH-2.5

# Status and Distribution

---

---

- Written in STELLA
  - Available in Lisp, C++ and Java
- Current release: PowerLoom 3.0.2.beta
  - Basic KR&R system only
  - Distributed as Lisp, C++ and Java source
  - ~500 downloads world-wide
  - ~400 subscribers to the mailing lists
- Licensing Terms
  - Open Source, user choice of 3 standard licences
    1. GPL
    2. LGPL
    3. Mozilla

# PowerLoom Features

- Full-function, robust and stable KR&R system
  - Representation, reasoning, query language, storage, extensive API
  - Available in Java (useful for integration with Protégé)
- Expressivity
  - KR failures often due to “we could not express X”
- Meta-representation & reasoning
  - Concepts, relations, contexts, rules, queries, etc. are all first-class citizens which can be represented and reasoned about
- Explanation support
  - Built-in recording and rendering of proof trees
  - Explanation of failed inferences (“WhyNot”)
- Sophisticated context & module system
  - Encapsulation and organization of knowledge
  - Efficient inference
  - Representation of assumptions: e.g., “all reactions modeled here assume 20°C ambient temperature”
- Sophisticated support for units & measures

# PowerLoom Features /2

---

---

- Extensible architecture
  - Easy to add new specialized reasoning procedures
- Scalability
  - Caveat: PowerLoom inference is worst-case exponential complexity
  - But: many design features to deal with performance
    - Common inferences (e.g. subsumption) supported by specialists
    - Expensive inference (e.g., classifier) available on demand
    - Various search control directives, e.g., forward/backward-only rules, resource bounded inference
    - Different inference levels
    - Modules to focus reasoning
    - Database interface to offload data-intensive operations onto RDBMS
  - Successfully handled very large KBs
    - Reasoned with full Cyc KB (~1,000,000 facts, 35,000 rules)
    - Large EELD ontologies and datasets (not loadable into XSB deductive database)  
 $O(1000)$  ontology & rules,  $O(10,000)$  instances,  $O(100,000)$  assertions (see example later)

---

---

# PowerLoom Language Concepts

# PowerLoom Representation Language

- PowerLoom language is based on KIF
  - The Knowledge Interchange Format (Genesereth 91)
  - Developed as part of DARPA's knowledge sharing effort
  - Proposed ANSI standard, now one of the accepted syntaxes of the Common-Logic effort
  - Syntax and declarative semantics for First-Order Predicate Logic
  - Lisp-based, uniform prefix syntax, similar to CycL
- Example:

Facts:   (person fred)  
             (citizen-of fred germany)  
             (national-language-of germany german)

Rules:   (forall (?p ?c ?l)  
             (=> (and (person ?p)  
                 (citizen-of ?p ?c)  
                 (national-language-of ?c ?l))  
                 (speaks-language ?p ?l))))

# PowerLoom Representation Language /2

---

- Many extensions to standard FOL:
  - Type, set & cardinality relations, e.g., `subset-of`, `instance-of`, `range-cardinality`, etc.
  - Second-order definitions via `holds`
  - Selective closed-world assumption (OWA is default)
  - Classical negation and negation-by-failure
  - Defaults (still need work)
- Frame-style definition language as syntactic sugar
  - `defconcept`, `defrelation`, `deffunction`, `definstance`, `defrule`
  - Allows concise definitions but expands internally into standard (more verbose) logic assertions

# PowerLoom Knowledge Bases

---

- Terminology Definitions
  - Concepts (classes), functions, and relations define the vocabulary of a domain, e.g., `person`, `citizen-of`, `age`, etc.
- Assertions
  - Describe what is true in a domain
  - Facts, e.g., `(person Fred)`
  - Rules, e.g., `(forall ?x (=> (rich ?x) (happy ?x)))`
- Contexts & Modules
  - Knowledge is organized into modules
  - Facts & rules are not asserted globally but relative to modules, can have different truth values in different modules
  - Hierarchical module structure, assertions from higher modules are inherited to lower modules

# Terms, Relations & Propositions

---

---

- A KB captures a useful representation of a physical or virtual world
- Entities in the world are modeled in the KB by ***terms***
  - “Georgia”, “Ben Franklin”, 3, “abc” , concept “Person”
- Terms are categorized and related via ***relations***
  - “has age”, “greater than”, “is married to”, “plus”, “Person”
  - Concepts such as “Person” are considered unary relations
- ***Propositions*** are sentences with an associated truth value
  - “Ben Franklin is a person”, “Bill is married to Hillary”, “two plus three equals six” (which is false)
- PowerLoom uses KIF terms and sentences to represent propositions
  - `(person Ben-Franklin) (married-to Bill Hillary)`  
`(= (+ 2 3) 6)`

# Logical Connectives & Rules

---

---

- Predicate logic uses ***logical connectives*** to construct complex sentences from simpler ones:
  - **and, or, not, <=, =>, <=>**, quantifiers **exists** and **forall**
- Examples:
  - “Richard is not a crook”:  
`(not (crook Richard))`
  - “Every person has a mother”:  
`(forall ?p  
 (=> (person ?p)  
 (exists ?m  
 (has-mother ?p ?m))))`

# Definitions

---

---

- Terminology (relations, concepts) need to be defined before they are used via `defconcept`, `deffunction` & `defrelation`
- Examples:

```
(defconcept person)
(defrelation married-to ((?p1 person) (?p2 person)))
(deffunction + ((?n1 number) (?n2 number))
  :-> (?sum number))
```

- Advantage & Disadvantage
  - Allows certain amount of error checking (e.g., misspelled relations, argument type violations)
  - A bit more tedious and can sometime generate ordering problems

# Definition Ordering

---



---

- Circular references are only allowed within definitions
  - Evaluation of rules within definitions is deferred until query time
- Example:

```
(defconcept parent (?p)
  :<=> (and (person ?p)
              (exists ?c (parent-of ?p ?c))))
(defrelation parent-of ((?p parent) (?c person)))
```

- Equivalent definition but illegal circular reference:

```
(defconcept parent)
(assert
  (forall (?p)
    (<=> (parent ?p)
            (and (person ?p) (exists ?c (parent-of ?p ?c))))))
(defrelation parent-of ((?p parent) (?c person)))
```

# Redefinitions

---

---

- Definition constructs primarily serve two roles
  1. Convenience; more compact syntax for often used idioms
  2. Linking sets of related axioms to a name to facilitate *redefinition*
- Redefinition is useful during interactive ontology and KB development
  
- Example Definition:

```
(defrelation parent-of ((?p person) (?c person))
  :<=> (relative-of ?p ?c))
```
- Example Redefinition:

```
(defrelation parent-of ((?p parent) (?c person)))
```
  
- Result:
  - Redefines **parent-of** with a different domain
  - Erases the rule ( $<=>$  (**parent-of** ?p ?c) (**relative-of** ?p ?c))

# Truth Values

---

- Each PowerLoom proposition (sentence) is associated with a truth value (relative to a context or module)
- Five possible truth values:
  - `true, false, default-true, default-false, unknown`
- Standard assertion assigns truth value `true`  
`(assert (person Bill))`
- Negation asserts truth value `false`  
`(assert (not (crook Richard)))`
- Presume command asserts default truth values  
`(presume (=> (bird ?x) (flies ?x)))`
- Propositions that are assigned `true` and `false` generate a clash (or contradiction)
  - Useful to detect certain constraint violations or errors
  - Used by proof-by-contradiction specialist
  - Contradictory propositions do not bring down the system and are treated as `unknown`

# Changing Truth Values

---

- The truth value of assertions can be changed
  - Implicitly, *by strengthening* the truth value, e.g., from default-true to true
  - By *explicit retraction* of the old truth value and new assertion, e.g.,  
`(assert (not (crook Richard)))  
(retract (not (crook Richard)))  
(assert (crook Richard))`
- Truth values of inferred propositions cannot be retracted

```
(defconcept employee (?e) :=> (person ?e))  
(assert (employee Mary))  
(ask (person Mary)) => true  
(retract (person Mary))  
(ask (person Mary)) => true
```

# Contexts & Modules

---

---

- Contexts & Modules
  - Knowledge is organized into contexts
    - Modules define name-spaces + assertion spaces
    - Worlds define assertion spaces only
  - Facts & rules are not asserted globally but relative to modules, can have different truth values in different modules
  - Hierarchical module structure, assertions from higher modules are inherited to lower modules
    - Non-monotonic inheritance is possible (e.g., override some inherited assertions for scenario reasoning)
  - Contexts are first-class objects that can be asserted to and queried about in the KB
    - Allows attachment of meta-information, e.g., source, assumptions, etc.
  - Very efficient, light-weight implementation derived from OPLAN
    - Support built in at a very low level (STELLA)

---

---

# An Annotated Example

# Using Modules

---



---

- We define a separate **BUSINESS** module for our example
  - Inherits built-in PowerLoom definitions from **PL-KERNEL/PL-USER**
  - Sets up a separate name and assertion space to avoid unwanted interference with/from other loaded knowledge bases
  - Allows easy experimentation (clearing/changing/editing/saving)
  - All PowerLoom commands are interpreted relative to current module

```
(defmodule "BUSINESS"
  :documentation "Module for the Business demo example."
  :includes ("PL-USER"))
```

**(in-module "BUSINESS")** ← List of inherited modules

**(clear-module "BUSINESS")** ← Set current module

Clear out local content

# Concepts

- Concepts define classes of entities
  - Defined via the `defconcept` command
  - Can have zero or more parent concepts (they all inherit `THING`)
  - Used to introduce typed instances

```
(defconcept company)
(defconcept corporation (?c company))
```

Simple “parentless” concept

```
(assert (company ACME-cleaners))
(assert (corporation megasoft))
```

Parent concept

```
(retrieve all ?x (company ?x))
There are 2 solutions:
#1: ?X=ACME-CLEANERS
#2: ?X=MEGASOFT
```

Concept variable (optional)

Create some instances

Retrieve all companies

```
(retrieve all ?x (corporation ?x))
There is 1 solution:
#1: ?X=MEGASOFT
```

Found via simple subsumption inference

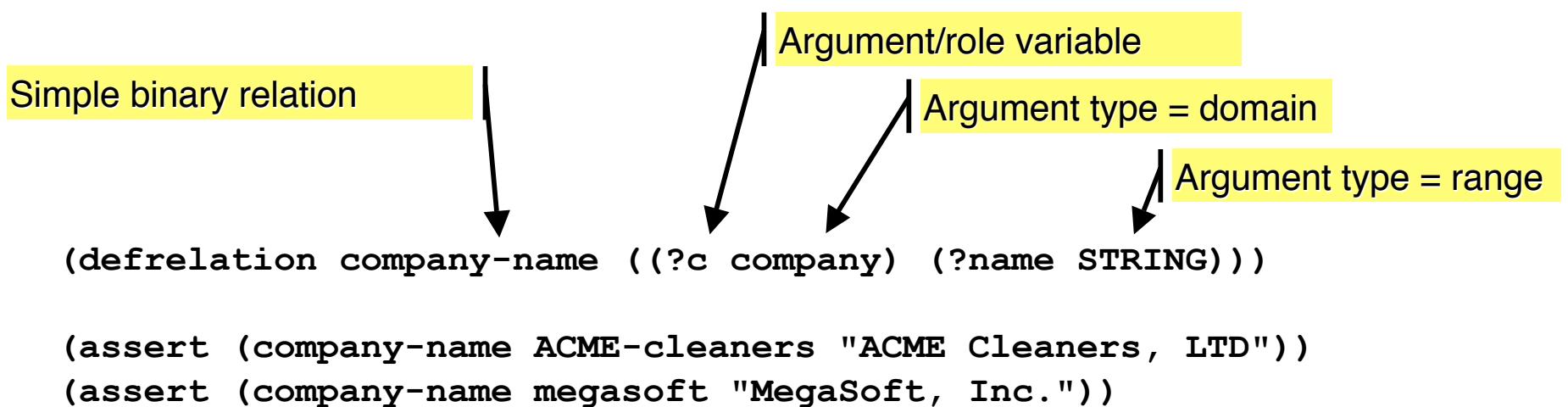
# Relations

---



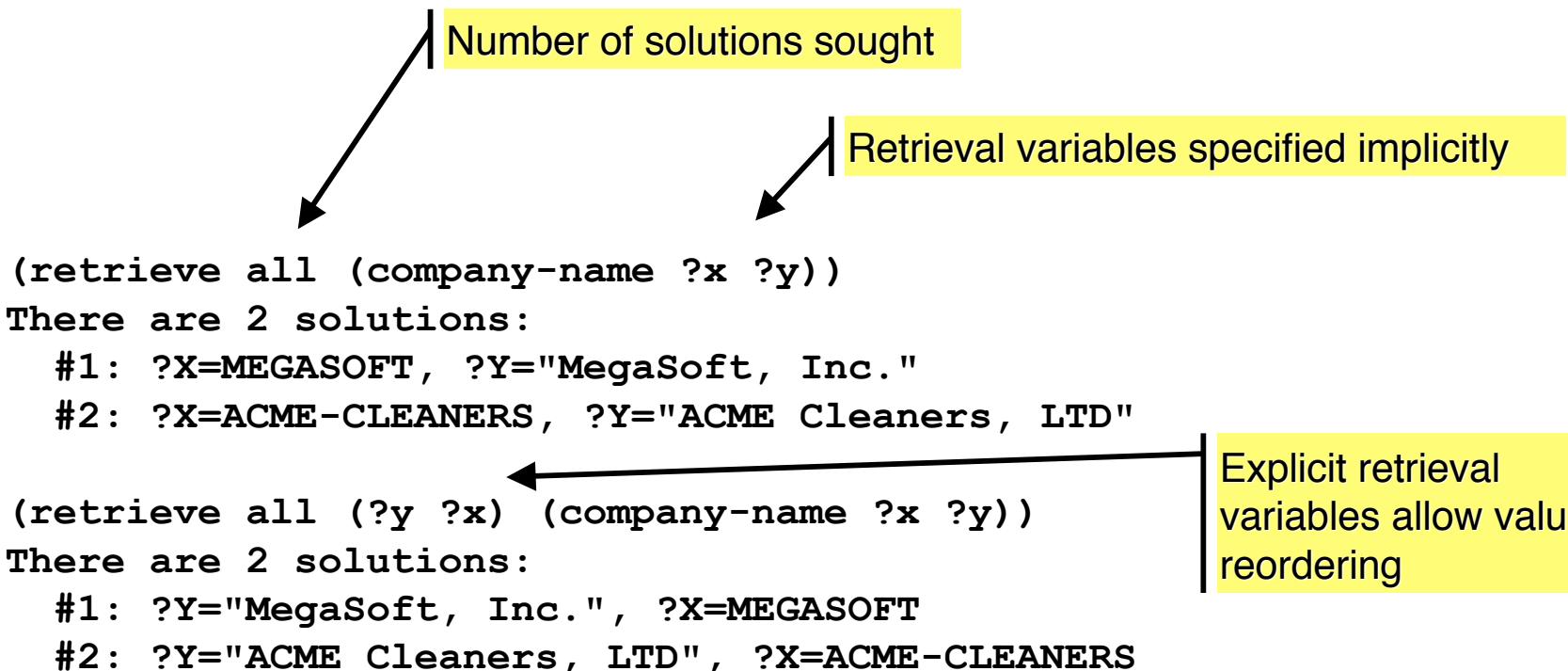
---

- Relations define sets of relationships between entities
  - Defined via the **defrelation** command (& **deffunction** see later)
  - Can have one or more arguments (unary to n-ary)
  - Can be fixed or variable arity
  - Can be single or multi-valued
  - Usually specify types for each argument
  - Used to specify relationships between entities



# Relations /2

- Retrieve all relations asserted in the BUSINESS module:



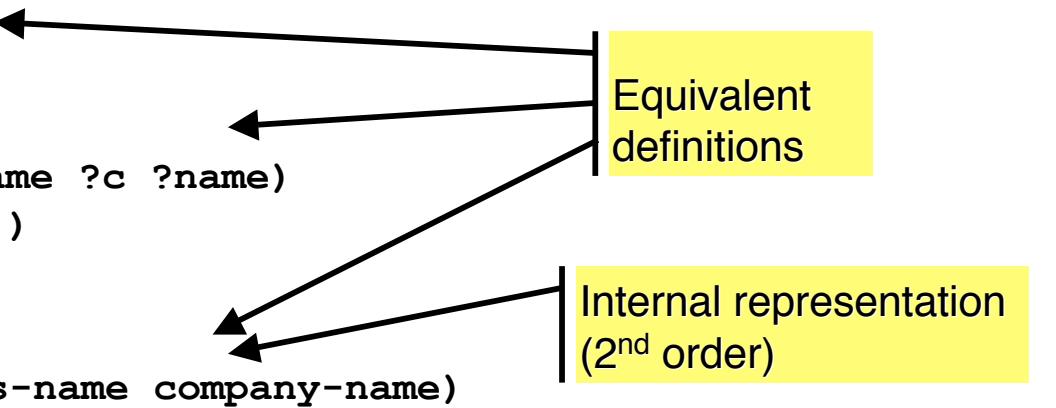
# Relation Hierarchies

- Hierarchies for concepts as well as relations are supported
  - PowerLoom represents a subconcept/subrelation relationship by asserting an “implication” relation (or an “implies” link)
  - Link is equivalent to a logic rule but allows more efficient inference
  - Various syntactic shortcuts are available to support often-used implication relations

```
(defrelation fictitious-business-name ((?c company) (?name STRING))
  :=> (company-name ?c ?name))
```

(forall (?c ?name)
 (=> (fictitious-business-name ?c ?name)
 (company-name ?c ?name)))

(subset-of fictitious-business-name company-name)



# Relation Hierarchies /2

---



---

- Retrieve all names of MegaSoft, fictitious or not
  - Illustrates that **company-name** is a multi-valued relation

```
(assert (fictitious-business-name megasoft "MegaSoft"))
```

```
(retrieve all ?x (company-name megasoft ?x))
```

There are 2 solutions:

```
#1: ?X="MegaSoft, Inc."
#2: ?X="MegaSoft"
```

Directly asserted

Inferred via the  
subrelation rule/link

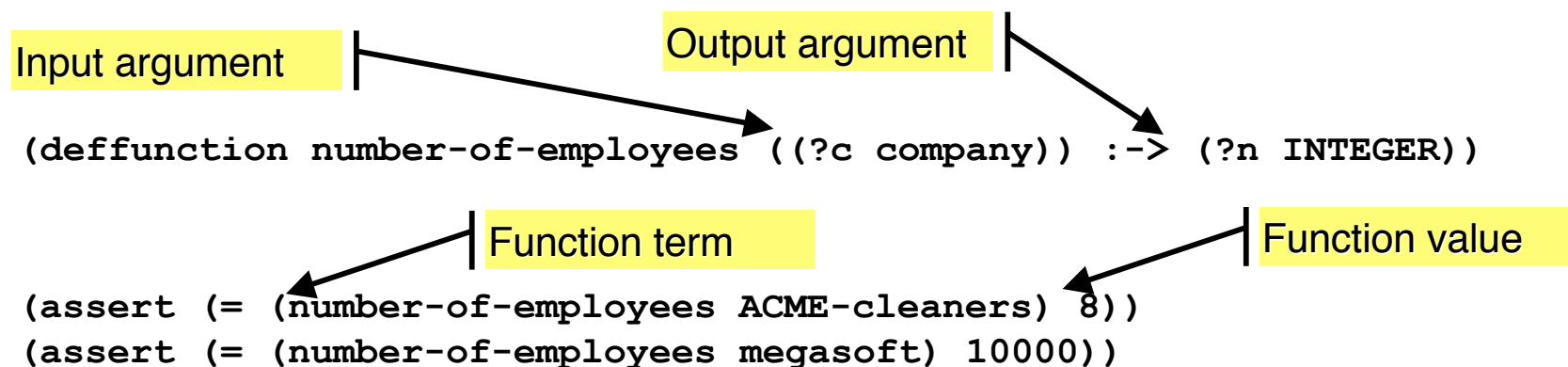
# Functions

---



---

- Functions are term-producing, single-valued relations
  - Defined via the **deffunction** command
  - Very similar to relations defined via **defrelation** but:
  - Term producing: a function applied to its first n-1 input arguments specifies a unique, intensional term, e.g., “Fred’s age”
  - Single-valued: each set of input arguments has at most one output argument (the last argument), e.g., “Fred’s age is 42”
  - By default, functions are assumed to be partial, i.e., could be undefined for some legal input values (e.g., 1/0)



# Functions /2

---



---

- Functions syntax often results in shorter expressions than using similar relation syntax:

```
(retrieve all (and (company ?x)
                    (< (number-of-employees ?x) 50)))
```

There is 1 solution:

#1: ?X=ACME-CLEANERS

- Compare to:

```
(retrieve all (and (company ?x)
                    (exists ?n
                            (and (number-of-employees ?x ?n)
                                 (< ?n 50)))))
```

There is 1 solution:

#1: ?X=ACME-CLEANERS

- Multiple function terms:

```
(retrieve all (> (number-of-employees ?x) (number-of-employees ?y)))
```

There is 1 solution:

#1: ?X=MEGASOFT, ?Y=ACME-CLEANERS

# Defined Concepts

---



---

- Concepts (and functions/relations) can be defined completely in terms of rules
- Useful to name often-used queries or subexpressions and build up powerful vocabulary

```
(defconcept small-company (?c company)
```

```
:<=> (and (company ?c)
```

```
< (number-of-employees ?c) 50)))
```

New keyword

Expands into these rules

```
(forall ?c (=> (and (company ?c)
                         (< (number-of-employees ?c) 50))
                         (small-company ?c)))
```

```
(forall ?c (=> (small-company ?c)
                     (and (company ?c)
                           (< (number-of-employees ?c) 50))))
```

## Defined Concepts /2

---



---

- Retrieve small companies even if we don't know exactly how many employees they have

```
(assert (and (company zz-productions)
             (< (number-of-employees zz-productions) 20)))
```

```
(retrieve all (small-company ?x))
```

There are 2 solutions:  
 #1: ?X=ZZ-PRODUCTIONS  
 #2: ?X=ACME-CLEANERS

Rule-based inference + transitivity of '<'

All we know is that ZZ Productions has less than 20 employees

# Negation & Open/Closed-World Semantics

- PowerLoom uses classical negation and an open-world assumption (OWA) by default
  - KB is not assumed to be a complete model of the world: if something can't be derived the answer is UNKNOWN, not FALSE
  - Can distinguish between failure and falsity!
  - Inference engine uses asymmetric effort to derive the truth or falsity of a query
    - Focuses effort on deriving truth, picks up falsity only via quick, shallow disproofs
    - Full effort for falsity available by asking for the negated query
    - Possible extension: 3-valued ask (similar to Loom)

```
(defconcept s-corporation ((?c corporation)))
```

```
(ask (s-corporation zz-productions)) => UNKNOWN
```

```
(ask (not (s-corporation zz-productions))) => UNKNOWN
```

Due to open-world assumption

```
(assert (not (s-corporation zz-productions)))
```

```
(ask (s-corporation zz-productions)) => FALSE
```

```
(ask (not (s-corporation zz-productions))) => TRUE
```

Quick disproof from assertion

# Negation & Open/Closed-World Semantics /2

- Falsity can also come from sources other than explicit assertion
  - Single-valued functions and relations
  - Inequalities
  - Disjoint types
  - Negated rule heads, etc.

```
(ask (= (number-of-employees ACME-cleaners) 8)) => TRUE
(ask (= (number-of-employees ACME-cleaners) 10)) => FALSE
(ask (not (= (number-of-employees ACME-cleaners) 10))) => TRUE
(ask (= (number-of-employees zz-productions) 100)) => FALSE
(ask (= (number-of-employees zz-productions) 10)) => UNKNOWN
```

Quick disproof since functions are single-valued

Quick disproof via inequality constraints

Truly unknown since there is not enough information

# Negation & Open/Closed-World Semantics /3

- Selective closed-world semantics and negation-by-failure are also available (as used by Prolog, deductive databases, F-Logic, etc.)
- Useful in cases where we do have complete knowledge
- If something can't be derived, it is assumed to be false
- Closed-world semantics specified by marking relations as `closed`
- Negation-by-failure via `fail` instead of `not`

```
(defrelation works-for (?p (?c company)))
```

```
(assert (works-for shirly ACME-cleaners))
(assert (works-for jerome zz-productions))
```

```
(ask (not (works-for jerome megasoft))) => UNKNOWN
```

Due to open world

```
(assert (closed works-for))
```

Mark relation as closed

```
(ask (not (works-for jerome megasoft))) => TRUE
```

Via selective closed-world semantics

```
(retract (closed works-for))
```

```
(ask (not (works-for jerome megasoft))) => UNKNOWN
```

```
(ask (fail (works-for jerome megasoft))) => TRUE
```

Via explicit negation-by-failure

# Retraction

---

- Retraction allows the erasure or change of a previously asserted truth-value of a proposition
  - Useful for error correction or iterative “change of mind” during development
  - Useful to change certain aspects of a scenario without having to reload the whole knowledge base
  - Allows efficient, fine-grained change
    - Some cached information is lost and needs to be regenerated
    - Loss can be minimized by careful structuring of module hierarchy (put more stable knowledge higher up in the hierarchy)
  - Allows the exploration of hypothetical conjectures
    - What would change if F were true or false?
    - Module system allows us to consider both possibilities at the same time

# Retraction /2

---

## ➤ Some geographic terminology and information

```
(defconcept geographic-location)
(defconcept country ((?1 geographic-location)))
(defconcept state ((?1 geographic-location)))
(defconcept city ((?1 geographic-location)))
(defrelation contains ((?11 geographic-location)
                      (?12 geographic-location)))

(assert (and
          (country united-states)
          (geographic-location eastern-us)
          (contains united-states eastern-us)
          (state georgia) (contains eastern-us georgia)
          (city atlanta) (contains georgia atlanta)
          (geographic-location southern-us)
          (contains united-states southern-us)
          (state texas) (contains eastern-us texas)
          (city dallas) (contains texas dallas)
          (city austin) (contains texas austin))))
```

# Retraction /3

---

---

- Retraction to fix an incorrect assertion

```
(ask (contains eastern-us texas)) => TRUE  
  
(retract (contains eastern-us texas))  
(assert (contains southern-us texas))  
  
(ask (contains eastern-us texas)) => UNKNOWN
```

# Value Clipping

---



---

- Functions allow implicit retraction via ***value clipping***
- Assertion of a function value automatically retracts a preexisting value
- Justified, since functions are single-valued

```
(deffunction headquarters ((?c company)) :-> (?city city))
```

```
(assert (= (headquarters zz-productions) atlanta))
(retrieve all (= ?x (headquarters zz-productions)))
```

There is 1 solution:

#1: ?X=ATLANTA

```
(assert (= (headquarters zz-productions) dallas))
(retrieve all (= ?x (headquarters zz-productions)))
```

There is 1 solution:

#1: ?X=DALLAS

Assertion automatically  
clips previous value

DALLAS value  
replaced ATLANTA

# Value Clipping /2

---

---

- Clipping also works for single-valued relations

```
(defrelation headquartered-in ((?c company) (?city city))
  :axioms (single-valued headquartered-in))
```

```
(assert (headquartered-in megasoft atlanta))
(retrieve all (headquartered-in megasoft ?x))
```

There is 1 solution:

```
#1: ?X=ATLANTA
```

```
(assert (headquartered-in megasoft dallas))
(retrieve all (headquartered-in megasoft ?x))
```

There is 1 solution:

```
#1: ?X=DALLAS
```

# Contradictions

---

---

- Propositions that are both **TRUE** and **FALSE** are contradictory
  - Contradictions can result from explicit assertions, during forward-chaining, or as the result of a refutation proof
  - Contradictory propositions are treated as **UNKNOWN** to allow the system to continue to function

```
(assert (not (state texas)))
```

Derived both **TRUE** and **FALSE** for the proposition `|P#|(STATE TEXAS)' .  
Clash occurred in module `|MDL|/PL-KERNEL-KB/BUSINESS' .

```
(ask (state texas)) => UNKNOWN  
(ask (not (state texas))) => UNKNOWN
```

# Rule-Based Inference

- Logic rules can be used to model complex relationships
  - Rules can be unnamed or named via `defrule`
  - Most definition commands expand into one or more rules
  - Inference engines apply rules to derive conclusions

```
(retrieve all (contains southern-us ?x))
```

There is 1 solution:

```
#1: ?X=TEXAS
```

Finds only directly asserted values

```
(defrule transitive-contains
  (forall (?11 ?12 ?13)
    (=> (and (contains ?11 ?12)
              (contains ?12 ?13))
         (contains ?11 ?13))))
```

Defines `contains` to be transitive

```
(defrule transitive-contains
  (=> (and (contains ?11 ?12)
            (contains ?12 ?13))
       (contains ?11 ?13)))
```

```
(retrieve all (contains southern-us ?x))
```

There are 3 solutions:

```
#1: ?X=TEXAS
#2: ?X=AUSTIN
#3: ?X=DALLAS
```

Same rule via implicit quantification

# Named Rules & Axiom Schemata

- Logic rules can be defined and named via `defrule`
  - Rules are propositions which are in the domain of discourse
    - Allows meta-annotations and reasoning
  - Naming rules (or any proposition) provides extra level of convenience
  - Axiom schemata allow simple definition of commonly used rule patterns

`(retract transitive-contains)`

`(retrieve all (contains southern-us ?x))`

There is 1 solution:

#1: ?X=TEXAS

`(assert (transitive contains))`

`(retrieve all (contains southern-us ?x))`

There are 3 solutions:

#1: ?X=TEXAS

#2: ?X=AUSTIN

#3: ?X=DALLAS

Retract rule by name

Reassert transitivity via meta-relation + axiom schema

```
(defrelation transitive ((?r RELATION))
  :=>> (and (binary-relation ?r)
               (not (function ?r)))
  :=>> (=> (and (?r ?x ?y)
                  (?r ?y ?z))
              (?r ?x ?z)))
```

Transitivity relation and axiom schema from PL-KERNEL KB

# Justifications and Explanation

---



---

- Explanation of true/false queries
  - Backward inference can store proof trees that can be rendered into explanations
  - Simple built-in explanation mechanism
    - Various rendering possibilities, ASCII, HTML, XML
    - Eliminates explanation of duplicate and low-level goals
    - Explanation strings for different audiences (technical, lay)

```
(ask (contains southern-us dallas)) => TRUE

(why)
1 (CONTAINS SOUTHERN-US DALLAS)
  follows by Modus Ponens
  with substitution {?11/SOUTHERN-US, ?13/DALLAS, ?12/TEXAS}
  since 1.1 ! (FORALL (?11 ?13)
    (<= (CONTAINS ?11 ?13)
        (EXISTS (?12)
          (AND (CONTAINS ?11 ?12)
              (CONTAINS ?12 ?13))))))
  and 1.2 ! (CONTAINS SOUTHERN-US TEXAS)
  and 1.3 ! (CONTAINS TEXAS DALLAS)
```

# Explanation /2

---



---

- Explanation of retrieved results
  - Separate explanation for each derived solution
  - why explains most recently retrieved solution

(retrieve 3 (contains southern-us ?x))

There are 3 solutions so far:

#1: ?X=DALLAS

#2: ?X=TEXAS

#3: ?X=AUSTIN

(why)

1 (CONTAINS SOUTHERN-US AUSTIN)

follows by Modus Ponens

with substitution {?11/SOUTHERN-US, ?13/AUSTIN, ?12/TEXAS}

since 1.1 ! (FORALL (?11 ?13)

    (<= (CONTAINS ?11 ?13)

    (EXISTS (?12)

        (AND (CONTAINS ?11 ?12)

        (CONTAINS ?12 ?13))))

and 1.2 ! (CONTAINS SOUTHERN-US TEXAS)

and 1.3 ! (CONTAINS TEXAS AUSTIN)

# Contexts & Modules

---

---

- Hypothetical or scenario reasoning can be achieved by
  - creating a new context which inherits existing set of facts and
  - allows the exploration of "assumptions".
- In this example, we show how certain inherited assertions can be retracted and changed

```
(defmodule "ALTERNATE-BUSINESS"
  :includes "BUSINESS")

(in-module "ALTERNATE-BUSINESS")

(assert (and (company web-phantoms)
              (company-name web-phantoms "Web Phantoms, Inc.")))

(retract (company-name megasoft "MegaSoft, Inc."))
(assert (company-name megasoft "MegaZorch, Inc."))
```

# Contexts & Modules /2

---



---

- The **ALTERNATE-BUSINESS** module
  - inherits all of the information of its parent module
  - is subject to the specific changes made in the local module.

(in-module "BUSINESS")

(retrieve all (company-name ?x ?y))

There are 3 solutions:

- #1: ?X=MEGASOFT, ?Y="MegaSoft, Inc."
- #2: ?X=ACME-CLEANERS, ?Y="ACME Cleaners, LTD"
- #3: ?X=MEGASOFT, ?Y="MegaSoft"

(in-module "ALTERNATE-BUSINESS")

(retrieve all (company-name ?x ?y))

There are 4 solutions:

- #1: ?X=MEGASOFT, ?Y="MegaZorch, Inc."
- #2: ?X=WEB-PHANTOMS, ?Y="Web Phantoms, Inc."
- #3: ?X=ACME-CLEANERS, ?Y="ACME Cleaners, LTD"
- #4: ?X=MEGASOFT, ?Y="MegaSoft"

Changed local assertion

New local assertion

From “fictitious business name” assertion

# Cross-Contextual Reasoning

---



---

- Normally queries operate in the current module.
- The IST (IS-TRUE) relation (J. McCarthy) allows us to query about the state of knowledge in other modules.
- This also allows cross-module inference by binding variables across forms
- Example: “find all companies whose names differ in the two modules”

```
(in-module "BUSINESS")
```

```
(retrieve all (ist alternate-business (company-name ?x ?y)))
```

There are 4 solutions:

```
#1: ?X=MEGASOFT, ?Y="MegaZorch, Inc."
```

```
#2: ?X=ALTERNATE-BUSINESS/WEB-PHANTOMS, ?Y="Web Phantoms, Inc."
```

```
#3: ?X=ACME-CLEANERS, ?Y="ACME Cleaners, LTD"
```

```
#4: ?X=MEGASOFT, ?Y="MegaSoft"
```

```
(retrieve all (and (ist business (company-name ?x ?y))
```

```
                  (fail (ist alternate-business (company-name ?x ?y)))))
```

There is 1 solution:

```
#1: ?X=MEGASOFT, ?Y="MegaSoft, Inc."
```

# RDBMS to PowerLoom Mapping

---



---

**Defining a PowerLoom database instance `edb1`:**

```
(DEFDB edb1
      :dsn "EELD_EDB17JUN03_COMPLIANT_PUBLIC_PL"
      :user "scott"  :host "blackcat.isi.edu")
```

**Defining a PowerLoom relation `EDB-Person`  
that maps onto the EDB table `Person`:**

```
(DEFTABLE EDB-Person edb1 "PERSON"
          (?ENTITYID ?REPORTID ?SOURCEID
           ?LASTNAME ?FIRSTNAME ?MIDDLENAME ?NICKNAME ?GENDER
           ?COUNTRYCITIZENSHIP ?AGE ?BIRTHLOCATION ?RESIDENCE))
```

**Defining a PowerLoom lifting axiom that maps the  
ontology concept `Person` onto `EDB-Person`:**

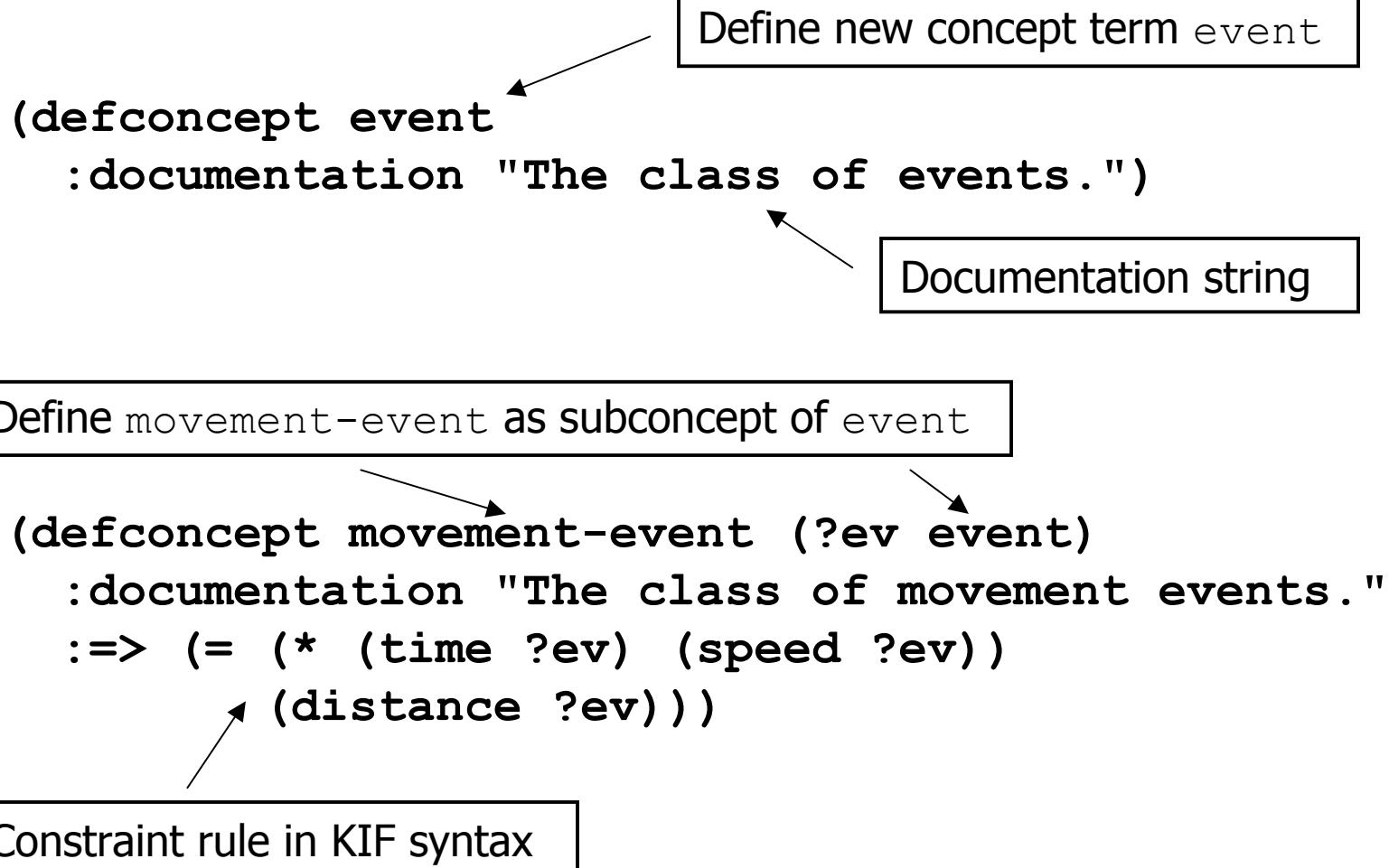
```
(ASSERT
  (=> (QUERY
        (EXISTS (?rep ?s ?l ?f ?m ?n ?g ?c ?a ?b ?res)
                (EDB-Person ?p ?rep ?s ?l ?f ?m ?n ?g ?c ?a ?b ?res))
        :MATCH-MODE :EELD :HOW-MANY :ALL)
        (Person ?p)))
```

---

---

# Advanced Topics

# Concept Definitions



# Definitions are Syntactic Sugar

- Constructs such as `defconcept` facilitate concise expression of commonly needed definition tasks
- Example: Previous definitions expand into the following more verbose set of assertions:

```
(defconcept event)
```

```
(defconcept movement-event)
```

```
(assert (documentation movement-event
                         "The class of movement events."))
```

```
(assert (forall ?ev
                (=> (movement-event ?ev)
                     (event ?ev))))
```

Meta assertion about the concept movement-event

```
(assert (forall ?ev
                (=> (movement-event ?ev)
                     (= (* (time ?ev) (speed ?ev))
                        (distance ?ev))))
```

Represents the subconcept relationship

Represents the constraint

# Relation Definitions

---



---

Define binary relation `sub-event` with domain and range `event`:

```
(defrelation sub-event ((?sub event) (?super event))
  :documentation "Links a sub-event to its super-event.")
```

Relations can have arbitrary as well as variable arity.

Functions are single-valued, term-generating relations:

```
(deffunction time ((?ev movement-event) ?time)
  :documentation "The duration of a movement event.")

(deffunction speed ((?ev movement-event) ?speed)
  :documentation "The speed of a movement event.")

(deffunction distance ((?ev movement-event) ?distance)
  :documentation "The distance covered by a movement.")
```

# Instance Definitions

Define event instance `ev1` with various properties:

```
(definstance ev1  
  :movement-event true :speed 10 :time 20)
```

The above concise, frame-style definition expands into the following individual assertions:

```
(assert (movement-event ev1))  
(assert (= (speed ev1) 10))  
(assert (= (time ev1) 20)))
```

Function term “time of event `ev1`”

# Attaching Information to Rules

---



---

- Rules are also in the domain of discourse and can be named for easy attachment of other information
  - For example, documentation strings or explanation templates:

```
(defrule speed-rule
  (forall ?ev
    (=> (movement-event ?ev)
         (= (* (time ?ev) (speed ?ev))
            (distance ?ev))))
  :documentation "Distance = time * speed of a movement.")

(assert
  (explanation-template speed-rule
    "The distance ?(distance ?ev) of the movement event ?ev equals
    its duration ?(time ?ev) times its speed ?(speed ?ev)."))
```

# Arithmetic Constraint Reasoning

---

- Arithmetic reasoning is important for scientific, engineering and everyday reasoning
- PoweLoom's built-in arithmetic specialists can compute a result from any two bound arguments
  - Allows us to model this formula via a single "speed" rule (instead of three – was an issue with Cyc in Phase-1)
  - Example:  $(+ 5 ?x 2) \Rightarrow ?x = -3$

```
(definstance ev1 :movement-event true :speed 10 :time 20)
(definstance ev2 :movement-event true :speed 10 :distance 50)

(retrieve all (distance ev1 ?x))
There is 1 solution:
#1: ?X=200

(retrieve all (time ev2 ?x))
There is 1 solution:
#1: ?X=5
```

# General Queries

---

---

- Many other systems have problems with general queries that ask about classes of things:
  - “Is it true that the ionization of diluted solutions is higher than those of concentrated solutions?”
  - This is often worked around by introducing a specific solution individual and asking the question about the instance
- In PowerLoom we can ask the universal question directly:

```
(defconcept solution :documentation "The class of chemical solutions.")  
(deffunction concentration-level ((?s solution) ?level)  
  :documentation "Concentration ?level of some particular solution ?s.")  
(deffunction ionization ((?s solution) ?level)  
  :documentation "Ionization ?level of some particular solution ?s.")  
(defrelation greater-than (?x ?y)  
  :documentation "Qualitative `>' relation.")
```

# General Queries /2

---



---

- Two (mock) rules describing relationships between concentration levels of solutions and their ionization level
  - Note that these rules operate at the instance level: given a specific solution instance and its concentration, we can infer the solution's ionization level

```
(defrule ionization-rule1
  (=> (and (solution ?x)
             (concentration-level ?x diluted))
       (= (ionization ?x) high))
  :documentation "Diluted solutions have high ionization.")

(defrule ionization-rule2
  (=> (and (solution ?x)
             (concentration-level ?x concentrated))
       (= (ionization ?x) low))
  :documentation "Concentrated solutions have low ionization.")

(assert (greater-than high low))
```

Qualitative ordering

# General Queries /3

---



---

- Can phrase the general query directly as a universally quantified statement
  - PowerLoom's Universal Introduction reasoner is used to prove it
  - Automatically introduces hypothetical solution individuals with the necessary properties in a hypothetical world

```
(ask (forall (?x ?y)
            (=> (and (solution ?x)
                      (solution ?y)
                      (concentration-level ?x diluted)
                      (concentration-level ?y concentrated))
                  (greater-than (ionization ?x) (ionization ?y))))))
⇒ TRUE
```

- Contrast this with the “hand-reification” approach:

```
(assert (and (solution sol1) (concentration-level sol1 diluted)))
(assert (and (solution sol2) (concentration-level sol2
concentrated)) )
```

```
(ask (greater-than (ionization sol1) (ionization sol2)))
⇒ TRUE
```

# Representing Queries

---



---

- Question answering applications
  - Reasoning about the queries itself is often important (e.g., answering a multiple-choice question by identifying the incorrect answers)
- PowerLoom can represent queries as terms to facilitate such query-level reasoning
  - Example: Wh-query can be represented via a **KAPPA** term and then evaluated via the query engine to generate the answers

```
(deffunction wh-query (?q) :-> (?kappa SET))

(assert (wh-query q1
  (kappa ?x
    (and (solution ?x) (concentration-level ?x diluted)))))

(assert (and (solution s1) (concentration-level s1 diluted)))
(assert (and (solution s2) (concentration-level s2 concentrated)))
(assert (and (solution s3) (concentration-level s3 diluted)))

(retrieve all ?x (member-of ?x (wh-query q1)))
There are 2 solutions:
#1: ?x=S3
#2: ?x=S1
```

# Units and Dimensions

---

---

- Scientific reasoning uses various units and dimensions
- PowerLoom has full support for units
  - Large number of predefined units
    - SI and other measurement systems
    - Fundamental quantities:  
mass, distance, time, angle, solid angle, amount of substance, electric current, luminous intensity, data
  - Arithmetic operations on units
  - Arbitrary combinations of units introduced by formulae
    - not limited to predefined combinations
  - Extensible via STELLA code
- Integration with Ontology
  - Datatype introduced via the **units** function
  - All logical operations and inferences work with units

# Reasoning with Units

## Assertions and Comparisons

- The **units** function introduces the data types

- Assertions
- Comparisons

```
(assert (= (age Fred) (units 35 "yr")))
(assert (= (age Pebbles) (units 18 "month")))
(assert (= (answer problem) (units 42 "m2kg/s3")))
```

Magnitude of expression  
Units (as string)

```
(ask (< (units 10 "mm") (units 10 "ft")))
TRUE
(ask (< (units 10 "ft") (units 11 "m")))
TRUE
(ask (< (units 11 "m") (units 10 "ft")))
FALSE
```

Arbitrary unit combinations

Comparisons normally give true or false answers

```
(ask (< (units 11 "kg") (units 10 "ft")))
UNKNOWN
```

Incompatible units, so no meaningful answer

# Reasoning with Units Conversions

- Conversions
  - All units are stored internally in canonical form (SI mks)
  - Conversions are performed on input or output

```
(retrieve (= (units ?x "mile") (units 100 "km"))))
```

There is 1 solution so far:

```
#1: ?X=62.13711922373341
```

Miles to kilometers

```
(retrieve (= (units ?x ?y) (units 100 "km"))))
```

There is 1 solution so far:

```
#1: ?X=100000.0, ?Y="m"
```

Both magnitude and unit  
(in canonical units)

```
(retrieve all (= (age Fred) (units ?x "yr"))))
```

There is 1 solution:

```
#1: ?X=35.0
```

More useful example

```
(retrieve (= (units 1000 ?y) (units 1 "km"))))
```

No solutions.

Too open-ended

# Reasoning with Units

## Arithmetic

- Arithmetic

- Units combine appropriately
- Arbitrary units combinations

```
(retrieve (= (units ?x ?y) (u-div (units 20 "km") (units 1 "h"))))
```

There is 1 solution so far:

```
#1: ?X=5.555555555555555, ?Y="m/s"
```

Creates 20 km/h unit,  
a common unit

Canonical internal  
representation.

```
(retrieve (= (units ?x "km/h") (u-div (units 20 "km") (units 1 "h"))))
```

There is 1 solution so far:

```
#1: ?X=20.0
```

Converted back to km/h

Synonyms

```
(retrieve (= (units ?x "km") (u* (units 20 "km/h") (units 1.5 "hr"))))
```

There is 1 solution so far:

```
#1: ?X=30.0
```

```
? (retrieve (= (units ?x ?y) (u-div (units 1 "h") (units 20 "km"))))
```

There is 1 solution so far:

```
#1: ?X=0.18, ?Y="s/m"
```

Creates 20h/km, a  
quite uncommon unit

# Time Points and Durations

---

- Time is an important aspect of the world
- PowerLoom has support for exact time points and durations
  - Time point specification uses flexible strings and `timepoint-of` function
    - ISO-8601 extended format for dates
    - Many other (US-centric) date formats supported:  
"5-Jan-2000", "1/5/2000", "January 10, 1997", "now", "today", "yesterday"
    - Time zones are specified numerically as offset from UTC  
(*i.e.*, what you **add** to UTC to get local time)  
Common time zone strings are also supported: UTC, Z, PST, EDT
  - Duration uses simple strings of days and milliseconds and `duration-of` function
    - "plus 5 days; 85000 ms", "minus 3 days; 0 ms"
    - Integrated and interchangeable with `units` function
  - Arithmetic operations on time points and durations
  - Comparisons of time points or durations
  - Integration with Ontology
    - Datatypes introduced via the `timepoint-of` and `duration-of` functions
    - All logical operations and inferences work with time points and durations
    - Durations interoperate with the `units` function

# Reasoning with Time Assertions and Comparisons

- The **timepoint-of** function introduces time points and the **duration-of** function introduces durations

- Assertions
- Comparisons

```
(assert (= (birthday Fred) (timepoint-of "2001-Jan-8 7:00Z")))  
(assert (= (duration Project85) (duration-of "180 days")))  
(assert (= (duration Concert75) (duration-of "0 days; 7200000 ms")))
```

```
(ask (< (timepoint-of "2005-Jul-3")  
       (timepoint-of "2005-Jul-4")))
```

TRUE

Type of time expression  
Specification (as string)

Comparisons normally give true or false answers

```
(ask (< (timepoint-of "2005-07-03T12:30Z")  
       (timepoint-of "2005-07-03T18:30+8:00")))
```

FALSE

Timezones are respected

```
(ask (< (timepoint-of "2006-10-May") (duration-of "2 days")))
```

UNKNOWN

Incompatible time types, so no meaningful answer

# Reasoning with Time Durations Conversions

- Conversions
  - Durations and time units can be converted between each other.
  - Time points can be destructured using the `timepoint-of*` function

```
(retrieve (= (duration-of ?x) (units 10 "day"))))
```

Simple conversion

There is 1 solution so far:

#1: ?X="plus 10 days; 0 ms"

```
(retrieve (= (duration-of ?x) (units 2 "h"))))
```

More useful example

There is 1 solution so far:

#1: ?X="plus 0 days; 7200000 ms"

```
(retrieve all (= (duration Concert75) (units ?x "h"))))
```

Convert to units

There is 1 solution:

#1: ?X=2.0

```
(retrieve all (= (duration Concert75) (units ?x ?y))))
```

Magnitude and unit  
(canonical units)

There is 1 solution:

#1: ?X=7200.0, ?Y="s"

```
(retrieve (= (timepoint-of* ?y ?m ?d ?hh ?mm ?ss "PST")
             (timepoint-of "Feb/5/2002 00:25:30 EST"))))
```

Destructuring  
with time zone  
conversion

There is 1 solution so far:

#1: ?Y=2002, ?M=2, ?D=4, ?HH=21, ?MM=25, ?SS=30.0

# Reasoning with Time Arithmetic

- Arithmetic
  - Addition and Subtraction of points and durations
  - Time types combine appropriately

```
(retrieve all (time- (timepoint-of "2006-10-20")
                     (timepoint-of "2006-10-15")
                     (duration-of ?interval)))
```

Amount of time between two time points

There is 1 solution:

```
#1: ?INTERVAL="plus 5 days; 0 ms"
(retrieve all (time+ (timepoint-of "2006-12-25")
                     (duration-of "12 days")
                     (timepoint-of ?date)))
```

Adding a duration to a time points to get a new time point

There is 1 solution:

```
#1: ?DATE="2007-JAN-06 7:00:00.000 UTC"
(retrieve all (time+ (duration-of "12 days")
                     (units ?n "h")
                     (duration-of "14 days")))
```

Works together with units

Output variable can be in any position

There is 1 solution:

```
#1: ?N=48.0
```

# Forward Inference (FIX)

---

---

- simple propositional reasoning, e.g.,  
(or p q) , ~p |= q
- simple equality reasoning
- forward skolemization, e.g.,  
(forall (?x Person)  
(exists (?y Person) (mother-of ?x ?y)))

---

---

# WhyNot Query Debugging

# Debugging Queries in Large KBs

- Logic-based knowledge representation & reasoning system
  - Use language of some logic L to represent knowledge (e.g., KIF)
  - Use implementation of proof procedure for L as reasoning engine
- Some (partially) developed knowledge base:

Facts:    (**person fred**)  
               (**citizen-of fred germany**)  
               (**national-language-of germany german**)

Rules:    (**forall (?p ?c ?l)**  
               (**=> (and (person ?p)**  
                   (**(citizen-of ?p ?c)**  
                   (**(national-language-of ?c ?l))**  
               (**speaks-language ?p ?l))))**

Queries:    (**speaks-language fred german**)?     $\Rightarrow$  **TRUE**  
               (**speaks-language fred french**)?     $\Rightarrow$  **UNKNOWN**

# The Problem

---

Failed query: `(speaks-language fred french) ?`  $\rightarrow$  UNKNOWN

Diagnosis is simple: the query failed because:

- Not asserted as a fact
- Not enough facts to infer it via the known rule
- Open world assumption!

Ask similar query in Cyc (1,000,000 facts, 35,000 rules)

- “Does Doug Lenat speak German?”
- Answer: UNKNOWN

Diagnosis is very difficult: the query failed because

- Not asserted as a fact
- All attempts to infer the answer failed
  - Search tree explored by Cyc is very large, timeout at 30 CPU seconds
- Hard to debug for knowledge engineers, impossible for non-experts

# Solution: Explaining Query Failures via Plausible Partial Proofs

- Standard explanation technique for logic-based reasoners:
  - Find and record a proof for the query
  - Present the proof to the user in an understandable way
  - **Problem: No proof  $\Rightarrow$  no explanation**
- **Solution:** Need to create a proof even though none could be found
- Generate plausible partial proofs for a query
  - Partial proofs can be explained
  - Proof "holes" identify potential knowledge or inference gaps
  - Multiple plausible partial proofs to explain different failure modes
  - Top-ranked partial proofs focus on most plausible failures
- Challenges:
  - What is a plausible partial proof?
  - Scaling, find plausible proofs without looking at too many options  
 $\Rightarrow$  there are infinitely many possible proofs!

# Example Explanation of Query Failure

```

Explanation #1 score=0.708:
1 (speaks-language fred german)
is true to some part because an if-then rule applies
with substitution {?p/fred, ?l/german, ?f/phil}
since 1.1 Rule: (forall (?p ?1)
  (<= (speaks-language ?p ?1)
    (exists (?f)
      (and (parent-of ?p ?f)
        (native-language-of ?f ?1)))))

and 1.2 Fact: (parent-of fred phil)
and 1.3 Inferred: (native-language-of phil german)

1.3 (native-language-of phil german)
is true to some part because an if-then rule applies
with substitution {?p/phil, ?l/german, ?c/germany}
since 1.3.1 Rule: (forall (?p ?1)
  (<= (native-language-of ?p ?1)
    (exists (?c)
      (and (person ?p)
        (birth-place-of ?p ?c)
        (national-language-of ?c ?1)))))

and 1.3.2 Fact: (person phil)
and 1.3.3 Unknown: (birth-place-of phil germany)
and 1.3.4 Fact: (national-language-of germany german)

```

# PowerLoom's “WhyNot” Query Diagnosis Tool

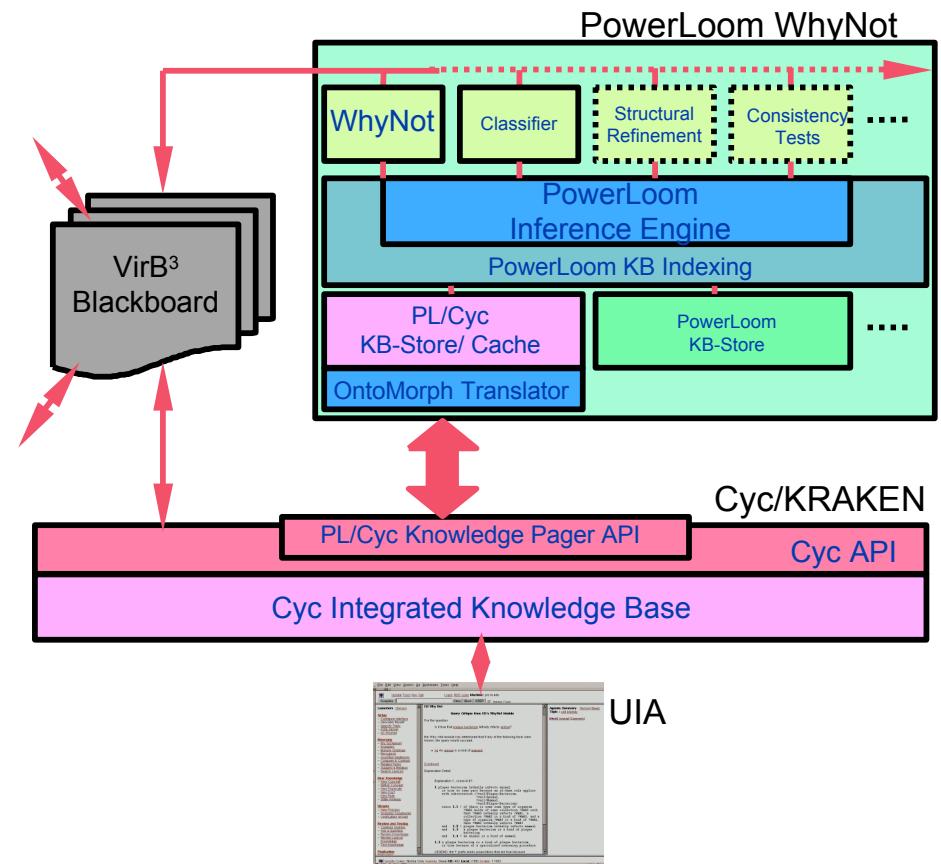
---

---

- PowerLoom KR&R system
  - First-order-logic-based KR&R system
  - Representation language is KIF (variant of FOL)
  - Natural deduction reasoner combining forward, backward reasoning plus variety of reasoning specialists
  - Type & cardinality reasoning, relation subsumption, classifier
  - Selective closed-world reasoning
  - Modules and light-weight worlds for hypothetical reasoning
- “WhyNot” built into inference engine of PowerLoom
  - Partial inference mode to generate plausible partial proofs
  - Score propagation instead of truth-values
  - Various plausibility heuristics
- PowerLoom explanation component used to explain partial proofs
- Only diagnosis of missing facts at the moment

# “WhyNot” Plug-in to Debug Queries in Large Cyc Knowledge Bases

- Cyc-based KRAKEN KA Tool
  - input and output in natural language
  - very large amount of background knowledge (over 1,000,000 facts, O(10,000) rules)
  - query diagnosis is very difficult
- PowerLoom “WhyNot”
  - external knowledge source integrated via blackboard
  - dynamically fetches and translates Cyc knowledge
  - performs partial inference against very large KB
  - pinpoints potential knowledge gaps
  - ships explanations to KRAKEN UIA display



REDHOUSE Cyc KB Browser - Netscape 6

File Edit View Search Go Bookmarks Tasks Help

<http://redhouse.isi.edu/cgi-bin/cyccgi/cg?cb-start>

Update Tools Nav Opt      Login: Lenat Machine: redhouse  
 Complete      Clear Show GREP  Ignore Case

[Refresh] [Reject] [Debug]

**ISI Why Not**

**Query Critique from ISI's WhyNot Module**

For the question

Does [Doug Lenat speak the German language?](#)

the Why-Not module has determined that if any of the following facts were known, the query would succeed.

- [\[+\] ?X speaks the German language.](#)
- [\[+\] Doug Lenat is a child of ?x.](#)

[\[Continue\]](#)

Explanation Detail:

Explanation 1, score=0.50:

1 [Doug Lenat speaks the German language.](#)  
 is true to some part because an if-then rule applies  
 with substitution {Y/[Doug Lenat](#), Z/[High German](#), X/{one of [Kurt Godel](#), [Georg Cantor](#), [Johann Sebastian Bach](#), etc.}}  
 since 1.1 Rule: [If Y is a child of X and X speaks Z, then Y speaks Z.](#)  
 and 1.2 Unknown: [Doug Lenat is a child of X.](#)  
 and 1.3 Fact: [X speaks the German language.](#)

**Legend:**

- The color **green** marks rules and facts that are known either because they were asserted or they follow by some logical inference.
- The color **yellow** marks logically inferred facts that are only partly true, since their inferences depend on some unknowns.
- The color **red** marks completely unknown facts that might be missing from the knowledge base (if they actually do make sense). Asserting all of them would allow Cyc to answer the

Say This

Agenda Summary [\[Refresh\]](#) [\[Reset\]](#) [\[My Thoughts\]](#) [\[Visualize\]](#)  
**Topic :** [cell biology](#)

[\[Next\]](#) [\[Journal\]](#)

**Essential Steps :**  
■ [Test Suite: WhyNot Queries](#) [\[UI 4\]](#)

**Relevant Suggestions :**  
■ [ISI Why-Not Proposal](#) [\[UI 8\]](#)

**Color Key**  
■ Can be done immediately  
■ Presently blocked

# Does Doug Lenat Speak German?

## Query Critique from ISI's WhyNot Module

For the question

Is it true that [Douglas Lenat](#) speaks [the German language](#)?

the Why-Not module has determined that if any of the following facts were known, the query would succeed.

- [+] [Douglas Lenat](#) is a child of [Kurt Godel](#).

Explanation 1, score=0.50:

- 1 [Douglas Lenat](#) speaks [the German language](#).

is true to some part because an if-then rule applies

with substitution { $Y/\text{Douglas Lenat}$ ,  $Z/\text{the German language}$ ,  $X/\text{Kurt Godel}$ }

since 1.1 ! If some [intelligent agent](#)  $Y$  is a child of some [intelligent agent](#)  $X$  and  $X$  speaks some [natural language](#), then  $Y$  speaks the [natural language](#).

and 1.2 ? [Douglas Lenat](#) is a child of [Kurt Godel](#).

and 1.3 ! [Kurt Godel](#) speaks [the German language](#).

Many similar explanations  
Need to generalize



# Improved Explanation by Generalizing Similar Proofs

For the question

Does [Doug Lenat speaks the German language?](#)

the Why-Not module has determined that if the following facts were known, the query would succeed.

- [\[+\] ?X speaks the German language.](#)
- [\[+\] Doug Lenat is a child of ?x.](#)

[\[Continue\]](#)

Explanation Detail:

Explanation 1, score=0.50:

1 [Doug Lenat speaks the German language.](#)

is true to some part because an if-then rule applies

with substitution {Y/[Doug Lenat](#), Z/[High German](#), X/{one of [Kurt Godel](#), [Georg Cantor](#),  
[Johann Sebastian Bach](#), etc.}}

since 1.1 **Rule:** [If Y is a child of X and X speaks Z, then Y speaks Z.](#)

and 1.2 **Unknown:** [Doug Lenat is a child of X.](#)

and 1.3 **Fact:** [X speaks the German language.](#)

# Alternative Lower-Score Explanation

---

---

➤ Explanation 2 (score 0.38):

For the question

Does Doug Lenat speak the German language?

the Why-Not module has determined that if the following facts were known, the query would succeed.

- [+] ?X speaks the German language.
- [+] Julius Caesar is a child of ?x.
- [+] Augustus and Doug Lenat are siblings.

# Alternative Lower-Score Explanation Detail

---



---

## ➤ Example Explanation 2 Detail (score 0.38)

Explanation 2, score=0.38:

- 1 [Doug Lenat speaks the German language.](#)

is true to some part because an if-then rule applies

with substitution {Y/[Doug Lenat](#), Z/[High German](#), X/[Julius Caesar](#).}

since 1.1 **Rule:** [If Y is a child of X and X speaks Z, then Y speaks Z.](#)

and 1.2 **Inferred:** [Doug Lenat is a child of Julius Caesar.](#)

and 1.3 **Inferred:** [Julius Caesar speaks the German language.](#)

- 1.2 [Doug Lenat is a child of Julius Caesar.](#)

is true to some part because an if-then rule applies

with substitution {Y/[Julius Caesar](#), Z/[Doug Lenat](#), X/[Augustus](#).}

since 1.2.1 **Rule:** [If X is a child of Y and X and Z are siblings, then Z is a child of Y.](#)

and 1.2.2 **Unknown:** [Augustus and Doug Lenat are siblings.](#)

and 1.2.3 **Fact:** [Augustus is a child of Julius Caesar.](#)

- 1.3 [Julius Caesar speaks the German language.](#)

is true to some part because an if-then rule applies

with substitution {Y/[Julius Caesar](#), Z/[High German](#), X/{one of [Kurt Godel](#), [Georg Cantor](#), [Johann Sebastian Bach](#), etc.}}

since 1.1 **Rule:** [If Y is a child of X and X speaks Z, then Y speaks Z.](#)

and 1.3.1 **Unknown:** [Julius Caesar is a child of X.](#)

and 1.3.2 **Fact:** [X speaks the German language.](#)

# Partial Inference Application #2: KOJAK Pattern Matching for Link Discovery (EELD)



- Link Discovery Problem
  - Given: large amounts of evidence
    - people, organizations, places, events, relationships, accounts, transactions, etc.
  - Discover: high-level activities of interest
    - Contract murders, gang wars, industry takeovers, terrorist activities, etc.
- KOJAK approach:
  - Represent evidence as large-scale PowerLoom evidence KBs
  - Represent domain knowledge via logic rules
  - Represent patterns via logic rules and queries
  - Use partial inference to detect patterns of interest
- Challenges:
  - Scale, incompleteness, noise, corruption

# Example: Using “WhyNot” Partial Inference in EELD Evaluation Domain

- Example domain (small/medium size):
  - 150 concepts, 200 relations, 6000 individuals
  - 10,000 asserted facts
  - 125 rules

- Example query:

(contract-murder UID517 UID3) ?

- Strict proof is fairly large:
  - 121 facts (or leaves)
  - 80 rule applications
  - Chaining depth up to 9
  - **Impossible to debug manually if it fails**
- Great domain for WhyNot partial match to show its utility:
  - Example: explain query failure caused by 1 missing assertion

Explanation #1 score=0.9966329966329965:

1. (**contract-kill** **UID517** **UID3**)  
 is true to some part because an if-then rule applies  
 with substitution (?mafia **UID517**, ?targetGroup **UID3**, ?installment9252, ?firstmpay21875, ?totalpay24709, ?city **Volgograd**, ?victin **UID524**)  
 since 1.1 Rule: (forall (?mafia ?targetGroup)  
 (=< ?contract-kill ?mafia ?targetGroup)  
 (exists (?installment ?firstmpay ?totalpay ?city ?region ?killer ?middleman1 ?middleman2 ?targets ?targetsGroup ?victin)  
 (=< ?contract-kill-method-019 ?victin ?middleman1 ?middleman2 ?targets ?targetsGroup ?victin))  
 and 1.2 Infewed: (**contract-kill-method-019** **UID524** **UID15** **UID535** **UID530** **UID3** **UID517** **UID531** **Region1** **Volgograd** 24709 21875 9252)

1.2. (**contract-kill-method-019** **UID524** **UID15** **UID535** **UID530** **UID3** **UID517** **UID531** **Region1** **Volgograd** 24709 21875 9252)  
 is true to some part because an if-then rule applies  
 with substitution (?victin **UID524**, ?victimGroup **UID15**, ?middleman1 **UID535**, ?middleman2 **UID530**, ?targetGroup **UID3**, ?mafia **UID517**, ?installment9252)  
 since 1.2.1 Rule: (forall (?victin ?middleman1 ?middleman2 ?targetsGroup ?mafia ?killer ?region ?city)  
 (=< ?contract-kill-method-019 ?victin ?middleman1 ?middleman2 ?targetsGroup ?mafia ?killer ?region ?city)  
 (and (?hasMembers ?targetsGroup ?victin)  
 (victin ?mafia ?victin)  
 (middleman1 ?mafia ?middleman1)  
 (middleman2 ?mafia ?middleman2)  
 (hisher ?mafia ?killer)  
 (operatesInRegion ?targetsGroup ?region)  
 (operatesInRegion ?region ?city)  
 (transfer-money ?victin ?middleman1 ?totalpay)  
 (planning ?middleman1 ?middleman2)  
 (transfer-money ?middleman1 ?middleman2 ?firstmpay)  
 (planning ?middleman1 ?middleman2)  
 (transfer-money ?middleman2 ?killer)  
 (transfer-money ?middleman2 ?killer ?installment)  
 (first-degree-murder ?killer ?victin ?city)  
 (ctrl-contact ?killer ?middleman2 **Report**)  
 (ctrl-contact ?middleman2 ?middleman1 **Report**)  
 (ctrl-contact ?middleman1 ?victin **Report**)  
 (record-c3 ?victin ?middleman1 ?middleman2)))  
 and 1.2.2 Fact: (**OperatesInRegion** **UID3** **Region1**)  
 and 1.2.3 Fact: (**victin** **UID517** **UID524**)  
 and 1.2.4 Fact: (**hasMembers** **UID3** **UID15**)  
 and 1.2.5 Fact: (**middleman** **UID517** **UID530**)  
 and 1.2.6 Fact: (**hisher** **UID517** **UID531**)  
 and 1.2.7 Infewed: (**Planning** **UID530** **UID531**)  
 and 1.2.8 Infewed: (**Ctrl-contact** **UID531** **UID530 Report**)  
 and 1.2.9 Fact: (**middleman** **UID517** **UID535**)  
 and 1.2.10 Infewed: (**Planning** **UID535** **UID530**)  
 and 1.2.11 Infewed: (**Planning** **UID524** **UID535**)  
 and 1.2.12 Infewed: (**Ctrl-contact** **UID530** **UID535 Report**)  
 and 1.2.13 Infewed: (**Ctrl-contact** **UID535** **UID524 Report**)  
 and 1.2.14 Infewed: (**Record-c3** **UID524** **UID15** **UID535** **UID530**)  
 and 1.2.15 Fact: (**GeographicalSubRegions** **Region1** **Volgograd**)  
 and 1.2.16 Infewed: (**First-degree-murder** **UID531** **UID15** **Volgograd**)  
 and 1.2.17 Infewed: (**Transfer-money** **UID535** **UID530** 21875)  
 and 1.2.18 Infewed: (**Transfer-money** **UID524** **UID535** 24709)  
 and 1.2.19 Infewed: (**Transfer-money** **UID530** **UID531** 9252)

1.2.7 (**Planning** **UID530** **UID531**)  
 is true because an if-then rule applies  
 with substitution (?X1 **UID530**, ?X2 **UID531**)  
 since 1.2.7.1 Rule: (forall (?X1 ?X2)  
 (=< (?planning ?X1 ?X2)  
 (?planning-method-021 ?X1 ?X2)))  
 and 1.2.7.2 Infewed: (**planning-method-021** **UID530** **UID531**)

1.2.7.2 (**planning-method-021** **UID530** **UID531**)  
 is true because an if-then rule applies  
 with substitution (?contractor **UID530**, ?contractee **UID531**)  
 since 1.2.7.2.1 Rule: (forall (?contractor ?contractee)  
 (=< (?planning-method-021 ?contractor ?contractee)  
 (?and (?ctrl-contact ?contractor ?contractee ?contractee **Propose**)  
 (?ctrl-contact ?contractor ?contractee ?contractor **Accept**)  
 (?ctrl-contact ?contractor ?contractee **Instruction**)  
 (?record-planning ?contractor ?contractee))))  
 and 1.2.7.2.2 Infewed: (**ctrl-contact** **UID530** **UID531** **Propose**)  
 and 1.2.7.2.3 Infewed: (**ctrl-contact** **UID531** **UID530** **Accept**)  
 and 1.2.7.2.4 Infewed: (**ctrl-contact** **UID530** **UID531** **Instruction**)  
 and 1.2.7.2.5 Infewed: (**Record-planning** **UID530** **UID531**)

1.2.7.2.2 (**ctrl-contact** **UID530** **UID531** **Propose**)  
 is true because an if-then rule applies  
 with substitution (?X1 **UID530**, ?X2 **UID531**, ?X3 **Propose**)  
 since 1.2.7.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
 (=< (?ctrl-contact ?X1 ?X2 ?X3)  
 (?ctrl-contact-method-010 ?X1 ?X2 ?X3)))  
 and 1.2.7.2.2.2 Infewed: (**ctrl-contact-method-010** **UID530** **UID531** **Propose**)

1.2.7.2.2.2 (**ctrl-contact-method-010** **UID530** **UID531** **Propose**)  
 is true because an if-then rule applies  
 with substitution (?X1 **UID530**, ?X2 **UID531**, ?X3 **Propose**)  
 since 1.2.7.2.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
 (=< (?contact-email ?X1 ?X2 ?X3)  
 (?contact-email-method-019 ?X1 ?X2 ?X3)))  
 and 1.2.7.2.2.2.2 Infewed: (**contact-email** **UID530** **UID531** **Propose**)

1.2.7.2.2.2.2 (**contact-email** **UID530** **UID531** **Propose**)  
 is true because an if-then rule applies  
 with substitution (?X1 **UID530**, ?X2 **UID531**, ?X3 **Propose**)  
 since 1.2.7.2.2.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
 (=< (?contact-email ?X1 ?X2 ?X3)  
 (?contact-email-method-029 ?X1 ?X2 ?X3)))  
 and 1.2.7.2.2.2.2.2 Infewed: (**contact-email-method-029** **UID530** **UID531** **Propose**)

1.2.7.2.2.2.2.2 (**contact-email-method-029** **UID530** **UID531** **Propose**)  
 is true because an if-then rule applies  
 with substitution (?X1 **UID530**, ?X2 **UID531**, ?X3 **Propose**)  
 since 1.2.7.2.2.2.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
 (=< (?contact-email-method-029 ?X1 ?X2 ?X3)  
 (?record-email ?X1 ?X2 ?X3)))  
 and 1.2.7.2.2.2.2.2.2 Infewed: (**Record-email** **UID530** **UID531** **Propose**)

1.2.7.2.2.2.2.2.2 (**Record-email** **UID530** **UID531** **Propose**)  
 is true because an if-then rule applies  
 with substitution (?from **UID530**, ?to **UID531**, ?content **Propose**, ?time2/5/2002, ?newid **UID6039**)  
 since 1.2.7.2.2.2.2.2.1 Rule: (forall (?from ?to ?content)  
 (=< (?record-email ?from ?to ?content)  
 (?exists (?from ?newid)  
 (?and (?mail-sending ?newid)  
 (?dateOfEvent ?newid ?time)  
 (?senderOfInfo ?newid ?from)  
 (?recipientOfInfo ?newid ?to)  
 (?telelocutionaryForce ?newid ?content))))))  
 and 1.2.7.2.2.2.2.2.2 Infewed: (**RecipientOfInfo** **UID6039** **UID531**)  
 and 1.2.7.2.2.2.2.2.3 Infewed: (**SenderOfInfo** **UID6039** **UID530**)  
 and 1.2.7.2.2.2.2.2.4 Infewed: (**Email-Sending** **UID6039**)  
 and 1.2.7.2.2.2.2.2.5 Infewed: (**TelelocutionaryForce** **UID6039** **Propose**)  
 and 1.2.7.2.2.2.2.2.6 Infewed: (**DateOfEvent** **UID6039** 2/5/2002)

1.2.7.2.3 (**ctrl-contact** **UID531** **UID530** **Accept**)  
 is true because an if-then rule applies  
 with substitution (?X1 **UID531**, ?X2 **UID530**, ?X3 **Accept**)  
 since 1.2.7.2.3.1 Rule: (forall (?X1 ?X2 ?X3)  
 (=< (?ctrl-contact ?X1 ?X2 ?X3)  
 (?ctrl-contact-method-003 ?X1 ?X2 ?X3)))  
 and 1.2.7.2.3.2 Infewed: (**ctrl-contact-method-009** **UID531** **UID530** **Accept**)

1.2.7.2.3.2 (**ctrl-contact-method-009** **UID531** **UID530** **Accept**)  
 is true because an if-then rule applies  
 with substitution (?X1 **UID531**, ?X2 **UID530**, ?X3 **Accept**)  
 since 1.2.7.2.3.2.1 Rule: (forall (?X1 ?X2 ?X3)  
 (=< (?ctrl-contact-method-009 ?X1 ?X2 ?X3)  
 (?contact-phone ?X1 ?X2 ?X3)))  
 and 1.2.7.2.3.2.2 Infewed: (**contact-phone** **UID531** **UID530** **Accept**)

1.2.7.2.3.2.2 (**contact-phone** **UID531** **UID530** **Accept**)  
 is true because an if-then rule applies  
 with substitution (?P1 **UID531**, ?P2 **UID530**, ?Content **Accept**, ?Pn1 **UID548**)  
 since 1.2.7.2.3.2.2.1 Rule: (forall (?P1 ?P2 ?Content)  
 (=< (?contact-phone ?P1 ?P2 ?Content)  
 (?exists (?P2 ?Pn1)  
 (?contact-phone-method-030 ?Pn1 ?P2 ?Content ?P1 ?P2)))  
 and 1.2.7.2.3.2.2.2 Infewed: (**contact-phone-method-030** **UID548** **UID551** **Accept** **UID531** **UID530**)

1.2.7.2.3.2.2.2 (**contact-phone-method-030** **UID548** **UID551** **Accept** **UID531** **UID530**)  
 is true because an if-then rule applies  
 with substitution (?Pn1 **UID548**, ?Pn2 **UID551**, ?Content **Accept**, ?P1 **UID531**, ?P2 **UID530**)  
 since 1.2.7.2.3.2.2.2.1 Rule: (forall (?Pn1 ?Pn2 ?Content ?P1 ?P2)  
 (=< (?ctrl-phone-method-030 ?Pn1 ?Pn2 ?Content ?P1 ?P2)  
 (?and (?agentPhoneNumber ?P1 ?Pn1)  
 (?agentPhoneNumber ?P2 ?Pn2)  
 (?record-phone-call ?Pn1 ?Pn2 ?Content))))  
 and 1.2.7.2.3.2.2.2.2 Infewed: (**AgentPhoneNumber** **UID531** **UID548**)  
 and 1.2.7.2.3.2.2.2.3 Infewed: (**AgentPhoneNumber** **UID530** **UID551**)  
 and 1.2.7.2.3.2.2.2.4 Infewed: (**Record-phone-call** **UID548** **UID551** **Accept**)

1.2.7.2.3.2.2.2.4 (record-phone-call UID548 UID551 Accept)  
is true because an if then rule applies  
with substitution (?from<sub>1</sub>?to ?content)  
since 1.2.7.2.3.2.2.2.4.1 Rule: (forall (?from ?to ?content)  
((= (record-phone-call ?from ?to ?content)  
(exists (?time ?newid)  
(and (MakingAPhoneCall ?newid)  
(dateOfEvent ?newid ?time)  
(callerNumber ?newid ?from)  
(receiverNumber ?newid ?to)  
(itellLocutionaryForce ?newid ?content))))))  
and 1.2.7.2.3.2.2.2.4.2 Fact: (callerNumber UID6041 UID548)  
and 1.2.7.2.3.2.2.2.4.3 Fact: (makingAPhoneCall UID6041)  
and 1.2.7.2.3.2.2.2.4.4 Fact: (receiverNumber UID6041 UID551)  
and 1.2.7.2.3.2.2.2.4.5 Fact: (itellLocutionaryForce UID6041 Accept)  
and 1.2.7.2.3.2.2.2.4.6 Fact: (dateOfEvent UID6041 2/7/2002)

1.2.7.2.4 (ch1-contact UID530 UID531 Instruction)  
is true because an if then rule applies  
with substitution (?X1|UID530, ?X2|UID531, ?X3|Instruction)  
since 1.2.7.2.4.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (ctrl-contact ?X1 ?X2 ?X3)  
(ctrl-contact-method-010 ?X1 ?X2 ?X3)))  
and 1.2.7.2.4.2 Infered: (ch1-contact-method-010 UID530 UID531 Instruction)

1.2.7.2.4.2.1 (ctrl-contact-method-010 UID530 UID531 Instruction)  
is true because an if then rule applies  
with substitution (?X1|UID530, ?X2|UID531, ?X3|Instruction)  
since 1.2.7.2.4.2.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (ctrl-contact-method-010 ?X1 ?X2 ?X3)  
(hold-meeting ?X1 ?X2 ?X3)))  
and 1.2.7.2.4.2.2 Infered: (hold-meeting UID530 UID531 Instruction)

1.2.7.2.4.2.2 (hold-meeting UID530 UID531 Instruction)  
is true because an if then rule applies  
with substitution (?X1|UID530, ?X2|UID531, ?X3|Instruction)  
since 1.2.7.2.4.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (hold-meeting ?X1 ?X2 ?X3)  
(hold-meeting-method-031 ?X1 ?X2 ?X3)))  
and 1.2.7.2.4.2.2.2 Infered: (hold-meeting-method-031 UID530 UID531 Instruction)

1.2.7.2.4.2.2.2 (hold-meeting-method-031 UID530 UID531 Instruction)  
is true because an if then rule applies  
with substitution (?X1|UID530, ?X2|UID531, ?X3|Instruction)  
since 1.2.7.2.4.2.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (hold-meeting-method-031 ?X1 ?X2 ?X3)  
(record-meeting ?X1 ?X2 ?X3)))  
and 1.2.7.2.4.2.2.2.2 Infered: (record-meeting UID530 UID531 Instruction)

1.2.7.2.4.2.2.2.2 (record-meeting UID530 UID531 Instruction)  
is true because an if then rule applies  
with substitution (?from|UID530, ?to|UID531, ?content|Instruction, ?time|2/12/2002, ?newid|UID6043)  
since 1.2.7.2.4.2.2.2.2.1 Rule: (forall (?from ?to ?content)  
((= (record-meeting ?from ?to ?content)  
(exists (?time ?newid)  
(and (MeetingTakingPlace ?newid)  
(dateOfEvent ?newid ?time)  
(socialParticipants ?newid ?from)  
(socialParticipants ?newid ?to)  
(senderOfInfo ?newid ?from)  
(recipientOfInfo ?newid ?to)  
(itellLocutionaryForce ?newid ?content))))))  
and 1.2.7.2.4.2.2.2.2.2 Fact: (socialParticipants UID6043 UID531)  
and 1.2.7.2.4.2.2.2.2.3 Fact: (socialParticipants UID6043 UID530)  
and 1.2.7.2.4.2.2.2.2.4 Fact: (MeetingTakingPlace UID6043)  
and 1.2.7.2.4.2.2.2.2.5 Fact: (senderOfInfo UID6043 UID530)  
and 1.2.7.2.4.2.2.2.2.6 Fact: (recipientOfInfo UID6043 UID531)  
and 1.2.7.2.4.2.2.2.7 Fact: (itellLocutionaryForce UID6043 Instruction)  
and 1.2.7.2.4.2.2.2.8 Fact: (dateOfEvent UID6043 2/12/2002)

1.2.7.2.5 (record-planning UID530 UID531)  
is true because an if then rule applies  
with substitution (?contractor|UID530, ?subordinate|UID531, ?newid|UID6044)  
since 1.2.7.2.5.1 Rule: (forall (?contractor ?subordinate)  
((= (record-planning ?contractor ?subordinate)  
(exists (?newid)  
(and (planningToDoSomething ?newid)  
(deliberateActors ?newid ?contractor)  
(deliberateActors ?newid ?subordinate))))))  
and 1.2.7.2.5.2 Fact: (deliberateActors UID6044 UID531)  
and 1.2.7.2.5.3 Fact: (deliberateActors UID6044 UID530)  
and 1.2.7.2.5.4 Fact: (PlanningToDoSomething UID6044)

1.2.8 (ch1-contact UID531 UID530 Report)  
is true because an if then rule applies  
with substitution (?X1|UID531, ?X2|UID530, ?X3|Report)  
since 1.2.7.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (ctrl-contact ?X1 ?X2 ?X3)  
(ctrl-contact-method-010 ?X1 ?X2 ?X3)))  
and 1.2.8.1 Infered: (ch1-contact-method-010 UID531 UID530 Report)

1.2.8.1 (ch1-contact-method-010 UID531 UID530 Report)  
is true because an if then rule applies  
with substitution (?X1|UID531, ?X2|UID530, ?X3|Report)  
since 1.2.7.2.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (ctrl-contact-method-010 ?X1 ?X2 ?X3)  
(contact-email ?X1 ?X2 ?X3)))  
and 1.2.8.1.1 Infered: (contact-email UID531 UID530 Report)

1.2.8.1.1 (contact-email UID531 UID530 Report)  
is true because an if then rule applies  
with substitution (?X1|UID531, ?X2|UID530, ?X3|Report)  
since 1.2.7.2.2.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (contact-email ?X1 ?X2 ?X3)  
(contact-email-method-029 ?X1 ?X2 ?X3)))  
and 1.2.8.1.1.1 Infered: (contact-email-method-029 UID531 UID530 Report)

1.2.8.1.1.1 (contact-email-method-029 UID531 UID530 Report)  
is true because an if then rule applies  
with substitution (?X1|UID531, ?X2|UID530, ?X3|Report)  
since 1.2.7.2.2.2.2.2.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (contact-email-method-029 ?X1 ?X2 ?X3)  
(record-email ?X1 ?X2 ?X3)))  
and 1.2.8.1.1.1.1 Infered: (record-email UID531 UID530 Report)

1.2.8.1.1.1.1 (record-email UID531 UID530 Report)  
is true because an if then rule applies  
with substitution (?from|UID531, ?to|UID530, ?content|Report, ?time|3/3/2002, ?newid|UID6053)  
since 1.2.7.2.2.2.2.2.2.1 Rule: (forall (?from ?to ?content)  
((= (record-email ?from ?to ?content)  
(exists (?time ?newid)  
(and (EmailIsSending ?newid)  
(dateOfEvent ?newid ?time)  
(senderOfInfo ?newid ?from)  
(recipientOfInfo ?newid ?to)  
(itellLocutionaryForce ?newid ?content))))))  
and 1.2.8.1.1.1.1.1 Fact: (senderOfInfo UID6053 UID531)  
and 1.2.8.1.1.1.1.2 Fact: (EmailIsSending UID6053)  
and 1.2.8.1.1.1.1.3 Fact: (recipientOfInfo UID6053 UID530)  
and 1.2.8.1.1.1.1.4 Fact: (itellLocutionaryForce UID6053 Report)  
and 1.2.8.1.1.1.1.5 Fact: (dateOfEvent UID6053 3/3/2002)

1.2.10 (planning UID535 UID530)  
is true because an if then rule applies  
with substitution (?X1|UID535, ?X2|UID530)  
since 1.2.7.1 Rule: (forall (?X1 ?X2)  
((= (planning ?X1 ?X2)  
(planning-method-021 ?X1 ?X2)))  
and 1.2.10.1 Infered: (planning-method-021 UID535 UID530)

1.2.10.1 (planning-method-021 UID535 UID530)  
is true because an if then rule applies  
with substitution (?contractor|UID535, ?contractee|UID530)  
since 1.2.7.2.1 Rule: (forall (?contractor ?contractee)  
((= (planning-method-021 ?contractor ?contractee)  
(and (ctrl-contact ?contractor ?contractee ?contractor\_Propose)  
(ctrl-contact ?contractee ?contractor ?contractor\_Accept)  
(ctrl-contact ?contractor ?contractee ?contractor ?contractee)))  
and 1.2.10.1.1 Infered: (ch1-contact UID535 UID530 Propose)  
and 1.2.10.1.2 Infered: (ch1-contact UID535 UID530 Accep)  
and 1.2.10.1.3 Infered: (ch1-contact UID535 UID530 Instruction)  
and 1.2.10.1.4 Infered: (record-planning UID535 UID530)

1.2.10.1.1 (ch1-contact UID535 UID530 Propose)  
is true because an if then rule applies  
with substitution (?X1|UID535, ?X2|UID530, ?X3|Propose)  
since 1.2.7.2.3.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (ctrl-contact ?X1 ?X2 ?X3)  
(ctrl-contact-method-009 ?X1 ?X2 ?X3)))  
and 1.2.10.1.1.1 Infered: (ch1-contact-method-009 UID535 UID530 Propose)

1.2.10.1.1.1 (ch1-contact-method-009 UID535 UID530 Propose)  
is true because an if then rule applies  
with substitution (?X1|UID535, ?X2|UID530, ?X3|Propose)  
since 1.2.7.2.3.2.1 Rule: (forall (?X1 ?X2 ?X3)  
((= (ctrl-contact-method-009 ?X1 ?X2 ?X3)  
(contact-phone ?X1 ?X2 ?X3)))  
and 1.2.10.1.1.1.1 Infered: (contact-phone UID535 UID530 Propose)

1.2.10.1.1.1.1 (`ch1-contact-method-009 UID535 UID530 Propose`)  
is true because an if then rule applies  
with substitution (?X1`UID535`, ?X2`UID530`, ?X3`(Propose)`)  
since 1.2.7.2.3.2.1 Rule: (forall (?x1 ?x2 ?x3)  
(<= (`ctrl-contact-method-009` ?x1 ?x2 ?x3)  
(`contact-phone` ?x1 ?x2 ?x3)))  
and 1.2.10.1.1.1.1 Infewed: (`Contact-phone UID535 UID530 Propose`)

1.2.10.1.1.1.1.1 (`contact-phone UID535 UID530 Propose`)  
is true because an if then rule applies  
with substitution (?P1`UID535`, ?P2`UID530`, ?Content`(Propose`, ?Pn2`UID551`, ?Pn1`UID530`)  
since 1.2.7.2.3.2.2.1 Rule: (forall (?P1 ?P2 ?Content)  
(<= (`contact-phone` ?P1 ?P2 ?Content)  
(exists (?Pn2 ?Pn1)  
(`contact-phone-method-030` ?Pn1 ?Pn2 ?Content ?P1 ?P2)))  
and 1.2.10.1.1.1.1.1 Infewed: (`contact-phone-method-030 UID536 UID551 Propose UID535 UID530`)

1.2.10.1.1.1.1.1.1 (`contact-phone-method-030 UID536 UID551 Propose UID535 UID530`)  
is true because an if then rule applies  
with substitution (?Pn1`UID536`, ?Pn2`UID551`, ?Content`(Propose`, ?P1`UID535`, ?P2`UID530`)  
since 1.2.7.2.3.2.2.2.1 Rule: (forall (?Pn1 ?Pn2 ?Content ?P1 ?P2)  
(<= (`contact-phone-method-030` ?Pn1 ?Pn2 ?Content ?P1 ?P2)  
(and (`agentPhoneNumbers` ?P1 ?Pn1)  
(`agentPhoneNumbers` ?P2 ?Pn2)  
(!`record-phone-call` ?Pn1 ?Pn2 ?Content))))  
and 1.2.10.1.1.1.1.1.1 Fact: (`agentPhoneNumber UID535 UID536`)  
and 1.2.7.2.3.2.2.3 Fact: (`agentPhoneNumber UID530 UID551`)  
and 1.2.10.1.1.1.1.1.2 Infewed: (`Record-phone-call UID536 UID551 Propose`)

1.2.10.1.1.1.1.2 (`Record-phone-call UID536 UID551 Propose`)  
is true because an if then rule applies  
with substitution (?From`UID536`, ?To`UID551`, ?Content`(Propose`, ?Time`1/25/2002`, ?Newid`UID6029`)  
since 1.2.7.2.3.2.2.2.4.1 Rule: (forall (?From ?To ?Content)  
(<= (!`record-phone-call` ?From ?To ?Content)  
(exists (?Time ?Newid)  
(and (`MakingAPhoneCall` ?Newid)  
(`DateOfEvent` ?Newid ?Time)  
(`callerNumber` ?Newid ?From)  
(`receiverNumber` ?Newid ?To)  
(`itelllocationaryForce` ?Newid ?Content))))  
and 1.2.10.1.1.1.1.2.1 Fact: (`callerNumber UID6029 UID536`)  
and 1.2.10.1.1.1.1.2.2 Fact: (`MakingAPhoneCall UID6029`)  
and 1.2.10.1.1.1.1.2.3 Fact: (`receiverNumber UID6029 UID551`)  
and 1.2.10.1.1.1.1.2.4 Fact: (`itelllocationaryForce UID6029 Propose`)  
and 1.2.10.1.1.1.1.2.5 Fact: (`DateOfEvent UID6029 1/25/2002`)

1.2.10.1.2 (`ch1-contact UID530 UID535 Accept`)  
is true because an if then rule applies  
with substitution (?X1`UID530`, ?X2`UID535`, ?X3`(Accept)`)  
since 1.2.7.2.3.1 Rule: (forall (?X1 ?X2 ?X3)  
(<= (`ctrl-contact` ?X1 ?X2 ?X3)  
(`ctrl-contact-method-009` ?X1 ?X2 ?X3)))  
and 1.2.10.1.2.1 Infewed: (`ch1-contact-method-009 UID530 UID535 Accept`)

1.2.10.1.2.1 (`ch1-contact-method-009 UID530 UID535 Accept`)  
is true because an if then rule applies  
with substitution (?X1`UID530`, ?X2`UID535`, ?X3`(Accept)`)  
since 1.2.7.2.3.2.1 Rule: (forall (?X1 ?X2 ?X3)  
(<= (`ctrl-contact-method-009` ?X1 ?X2 ?X3)  
(`contact-phone` ?X1 ?X2 ?X3)))  
and 1.2.10.1.2.1.1 Infewed: (`Contact-phone UID530 UID535 Accept`)

1.2.10.1.2.1.1 (`Contact-phone UID530 UID535 Accept`)  
is true because an if then rule applies  
with substitution (?P1`UID530`, ?P2`UID535`, ?Content`(Accept`, ?Pn2`UID536`, ?Pn1`UID551`)  
since 1.2.7.2.3.2.2.1 Rule: (forall (?P1 ?P2 ?Content)  
(<= (`contact-phone` ?P1 ?P2 ?Content)  
(exists (?Pn2 ?Pn1)  
(`contact-phone-method-030` ?Pn1 ?Pn2 ?Content ?P1 ?P2)))  
and 1.2.10.1.2.1.1.1 Infewed: (`Contact-phone-method-030 UID551 UID536 Accept UID530 UID535`)

1.2.10.1.2.1.1.1 (`Contact-phone-method-030 UID551 UID536 Accept UID530 UID535`)  
is true because an if then rule applies  
with substitution (?Pn1`UID551`, ?Pn2`UID536`, ?Content`(Accept`, ?P1`UID530`, ?P2`UID535`)  
since 1.2.7.2.3.2.2.2.1 Rule: (forall (?Pn1 ?Pn2 ?Content ?P1 ?P2)  
(<= (`contact-phone-method-030` ?Pn1 ?Pn2 ?Content ?P1 ?P2)  
(and (`agentPhoneNumbers` ?P1 ?Pn1)  
(`agentPhoneNumbers` ?P2 ?Pn2)  
(!`record-phone-call` ?Pn1 ?Pn2 ?Content))))  
and 1.2.7.2.3.2.2.3 Fact: (`agentPhoneNumber UID530 UID551`)  
and 1.2.10.1.1.1.1.1.1 Fact: (`agentPhoneNumber UID535 UID536`)  
and 1.2.10.1.1.1.1.1.1.1 Infewed: (`Record-phone-call UID551 UID536 Accept`)

1.2.10.1.2.1.1.1.1 (`Record-phone-call UID551 UID536 Accept`)  
is true because an if then rule applies  
with substitution (?From`UID551`, ?To`UID536`, ?Content`(Accept`, ?Time`1/27/2002`, ?Newid`UID6031`)  
since 1.2.7.2.3.2.2.2.4.1 Rule: (forall (?From ?To ?Content)  
(<= (!`record-phone-call` ?From ?To ?Content)  
(exists (?Time ?Newid)  
(and (`MakingAPhoneCall` ?Newid)  
(`DateOfEvent` ?Newid ?Time)  
(`callerNumber` ?Newid ?From)  
(`receiverNumber` ?Newid ?To)  
(`itelllocationaryForce` ?Newid ?Content))))  
and 1.2.10.1.2.1.1.1.1.1 Fact: (`callerNumber UID6031 UID551`)  
and 1.2.10.1.2.1.1.1.2 Fact: (`MakingAPhoneCall UID6031`)  
and 1.2.10.1.2.1.1.1.3 Fact: (`receiverNumber UID6031 UID536`)  
and 1.2.10.1.2.1.1.1.4 Fact: (`itelllocationaryForce UID6031 Accept`)  
and 1.2.10.1.2.1.1.1.5 Fact: (`DateOfEvent UID6031 1/27/2002`)

1.2.10.1.3 (`ch1-contact UID535 UID530 Instruction`)  
is true because an if then rule applies  
with substitution (?X1`UID535`, ?X2`UID530`, ?X3`(Instruction)`)  
since 1.2.7.2.3.1 Rule: (forall (?X1 ?X2 ?X3)  
(<= (`ctrl-contact` ?X1 ?X2 ?X3)  
(`ctrl-contact-method-009` ?X1 ?X2 ?X3)))  
and 1.2.10.1.3.1 Infewed: (`ch1-contact-method-009 UID535 UID530 Instruction`)

1.2.10.1.3.1 (`ch1-contact-method-009 UID535 UID530 Instruction`)  
is true because an if then rule applies  
with substitution (?X1`UID535`, ?X2`UID530`, ?X3`(Instruction)`)  
since 1.2.7.2.3.2.1 Rule: (forall (?X1 ?X2 ?X3)  
(<= (`ctrl-contact-method-009` ?X1 ?X2 ?X3)  
(`contact-phone` ?X1 ?X2 ?X3)))  
and 1.2.10.1.3.1.1 Infewed: (`Contact-phone UID535 UID530 Instruction`)

1.2.10.1.3.1.1 (`Contact-phone UID535 UID530 Instruction`)  
is true because an if then rule applies  
with substitution (?P1`UID535`, ?P2`UID530`, ?Content`(Instruction`, ?Pn2`UID551`, ?Pn1`UID536`)  
since 1.2.7.2.3.2.2.1 Rule: (forall (?P1 ?P2 ?Content)  
(<= (`contact-phone` ?P1 ?P2 ?Content)  
(exists (?Pn2 ?Pn1)  
(`contact-phone-method-030` ?Pn1 ?Pn2 ?Content ?P1 ?P2)))  
and 1.2.10.1.3.1.1.1 Infewed: (`Contact-phone-method-030 UID536 UID551 Instruction UID535 UID530`)

1.2.10.1.3.1.1.1 (`Contact-phone-method-030 UID536 UID551 Instruction UID535 UID530`)  
is true because an if then rule applies  
with substitution (?Pn1`UID536`, ?Pn2`UID551`, ?Content`(Instruction`, ?P1`UID535`, ?P2`UID530`)  
since 1.2.7.2.3.2.2.2.1 Rule: (forall (?Pn1 ?Pn2 ?Content ?P1 ?P2)  
(<= (`contact-phone-method-030` ?Pn1 ?Pn2 ?Content ?P1 ?P2)  
(and (`agentPhoneNumbers` ?P1 ?Pn1)  
(`agentPhoneNumbers` ?P2 ?Pn2)  
(!`record-phone-call` ?Pn1 ?Pn2 ?Content))))  
and 1.2.10.1.1.1.1.1 Fact: (`agentPhoneNumber UID535 UID536`)  
and 1.2.7.2.3.2.2.3 Fact: (`agentPhoneNumber UID530 UID551`)  
and 1.2.10.1.3.1.1.1 Infewed: (`Record-phone-call UID536 UID551 Instruction`)

1.2.10.1.3.1.1.1.1 (`Record-phone-call UID536 UID551 Instruction`)  
is true because an if then rule applies  
with substitution (?From`UID536`, ?To`UID551`, ?Content`(Instruction`, ?Time`1/30/2002`, ?Newid`UID6033`)  
since 1.2.7.2.3.2.2.2.4.1 Rule: (forall (?From ?To ?Content)  
(<= (!`record-phone-call` ?From ?To ?Content)  
(exists (?Time ?Newid)  
(and (`MakingAPhoneCall` ?Newid)  
(`DateOfEvent` ?Newid ?Time)  
(`callerNumber` ?Newid ?From)  
(`receiverNumber` ?Newid ?To)  
(`itelllocationaryForce` ?Newid ?Content))))  
and 1.2.10.1.3.1.1.1.1 Fact: (`callerNumber UID6033 UID536`)  
and 1.2.10.1.3.1.1.1.2 Fact: (`MakingAPhoneCall UID6033`)  
and 1.2.10.1.3.1.1.1.3 Fact: (`receiverNumber UID6033 UID551`)  
and 1.2.10.1.3.1.1.1.4 Fact: (`itelllocationaryForce UID6033 Instruction`)  
and 1.2.10.1.3.1.1.1.5 Fact: (`DateOfEvent UID6033 1/30/2002`)

1.2.10.1.4 (`Record-planning UID535 UID530`)  
is true because an if then rule applies  
with substitution (?Contractor`UID535`, ?Subordinate`UID530`, ?Newid`UID6035`)  
since 1.2.7.2.5.1 Rule: (forall (?Contractor ?Subordinate)  
(<= (!`Record-planning` ?Contractor ?Subordinate)  
(exists (?Newid)  
(and (`PlanningToSomething` ?Newid)  
(`DeliberateActors` ?Newid ?Contractor)  
(`DeliberateActors` ?Newid ?Subordinate))))  
and 1.2.10.1.4.1 Fact: (`DeliberateActors UID6035 UID535`)  
and 1.2.10.1.4.2 Fact: (`PlanningToDoSomething UID6035`)  
and 1.2.10.1.4.3 Fact: (`DeliberateActors UID6035 UID530`)

1.2.11 (planning UID524 UID535)  
is true because an if-then rule applies  
with substitution (?X1<sub>UID524</sub>, ?X2<sub>UID535</sub>)  
since 1.2.11.1 Rule: (forall (?x1 ?x2)  
(<= (planning ?x1 ?x2)  
(planning-method-023 ?x1 ?x2)))  
and 1.2.11.2 Inferred: (planning-method-023 UID524 UID535)

1.2.11.2 (planning-method-023 UID524 UID535)  
is true because an if-then rule applies  
with substitution (?contractor<sub>UID524</sub>, ?contractee<sub>UID535</sub>)  
since 1.2.11.2.1 Rule: (forall (?contractor ?contractee)  
(<= (planning-method-023 ?contractor ?contractee)  
(and (ctrl-contact ?contractor ?contractee instruction)  
(record-planning ?contractor ?contractee))))  
and 1.2.11.2.2 Inferred: (ctrl-contact UID524 UID535 instruction)  
and 1.2.11.2.3 Inferred: (record-planning UID524 UID535)

1.2.11.2.2 (ctrl-contact UID524 UID535 instruction)  
is true because an if-then rule applies  
with substitution (?X1<sub>UID524</sub>, ?X2<sub>UID535</sub>, ?X3<sub>instruction</sub>)  
since 1.2.7.2.3.1 Rule: (forall (?x1 ?x2 ?x3)  
(<= (ctrl-contact-method-003 ?x1 ?x2 ?x3)  
(ctrl-contact-method-003 ?x1 ?x2 ?x3)))  
and 1.2.11.2.2.1 Inferred: (ctrl-contact-method-009 UID524 UID535 instruction)

1.2.11.2.2.1 (ctrl-contact-method-009 UID524 UID535 instruction)  
is true because an if-then rule applies  
with substitution (?X1<sub>UID524</sub>, ?X2<sub>UID535</sub>, ?X3<sub>instruction</sub>)  
since 1.2.7.2.3.2.1 Rule: (forall (?x1 ?x2 ?x3)  
(<= (ctrl-contact-method-003 ?x1 ?x2 ?x3)  
(contact-phone ?x1 ?x2 ?x3)))  
and 1.2.11.2.2.1.1 Inferred: (contact-phone UID524 UID535 instruction)

1.2.11.2.2.1.1 (contact-phone UID524 UID535 instruction)  
is true because an if-then rule applies  
with substitution (?p1<sub>UID524</sub>, ?p2<sub>UID535</sub>, ?content<sub>instruction</sub>, ?pn1<sub>UID536</sub>, ?pn2<sub>UID529</sub>)  
since 1.2.7.2.3.2.2.1 Rule: (forall (?p1 ?p2 ?content)  
(<= (contact-phone ?p1 ?p2 ?content)  
(exists (?pn2 ?pn1)  
(contact-phone-method-030 ?pn1 ?pn2 ?content ?p1 ?p2)))  
and 1.2.11.2.2.1.1.1 Inferred: (record-phone-call UID525 UID536 instruction UID524 UID535)

1.2.11.2.2.1.1.1 (record-phone-call UID525 UID536 instruction UID524 UID535)  
is true because an if-then rule applies  
with substitution (?pn1<sub>UID525</sub>, ?pn2<sub>UID536</sub>, ?content<sub>instruction</sub>, ?p1<sub>UID524</sub>, ?p2<sub>UID535</sub>)  
since 1.2.7.2.3.2.2.2.1 Rule: (forall (?pn1 ?pn2 ?content ?p1 ?p2)  
(<= (contact-phone-method-030 ?pn1 ?pn2 ?content ?p1 ?p2)  
(and (agentPhoneNumber ?p1 ?pn1)  
(agentPhoneNumber ?p2 ?pn2)  
(!record-phone-call ?pn1 ?pn2 ?content))))  
and 1.2.11.2.2.1.1.1.1 Fact: (agentPhoneNumber UID524 UID525)  
and 1.2.11.2.2.1.1.1.2 Fact: (agentPhoneNumber UID535 UID536)  
and 1.2.11.2.2.1.1.1.3 Inferred: (record-phone-call UID525 UID536 instruction)

1.2.11.2.2.1.1.2 (record-phone-call UID525 UID536 instruction)  
is true because an if-then rule applies  
with substitution (?from<sub>UID525</sub>, ?to<sub>UID536</sub>, ?content<sub>instruction</sub>, ?time/1/19/2002, ?newid<sub>UID6024</sub>)  
since 1.2.7.2.3.2.2.2.4.1 Rule: (forall (?from ?to ?content)  
(<= (!record-phone-call ?from ?to ?content)  
(exists (?time ?newid)  
(and (MakingAPhoneCall ?newid)  
(dateOfEvent ?newid ?time)  
(callerNumber ?newid ?from)  
(receiverNumber ?newid ?to)  
(itellocationaryForce ?newid ?content))))  
and 1.2.11.2.2.1.1.1.2.1 Fact: (callerNumber UID6024 UID525)  
and 1.2.11.2.2.1.1.1.2.2 Fact: (MakingAPhoneCall UID6024)  
and 1.2.11.2.2.1.1.1.2.3 Fact: (receiverNumber UID6024 UID536)  
and 1.2.11.2.2.1.1.1.2.4 Fact: (itellocationaryForce UID6024 instruction)  
and 1.2.11.2.2.1.1.1.2.5 Fact: (dateOfEvent UID6024 1/19/2002)

1.2.11.2.3 (record-planning UID524 UID535)  
is true because an if-then rule applies  
with substitution (?contractor<sub>UID524</sub>, ?subordinate<sub>UID535</sub>, ?newid<sub>UID6026</sub>)  
since 1.2.7.2.5.1 Rule: (forall (?contractor ?subordinate)  
(<= (record-planning ?contractor ?subordinate)  
(exists (?newid)  
(and (deliberateActors ?newid)  
(deliberateActors ?newid ?contractor)  
(deliberateActors ?newid ?subordinate))))  
and 1.2.11.2.3.1 Fact: (deliberateActors UID6026 UID524)  
and 1.2.11.2.3.2 Fact: (PlanningToDoSomething UID6026)  
and 1.2.11.2.3.3 Fact: (deliberateActors UID6026 UID535)

1.2.12 (ctrl-contact UID530 UID535 Report)  
is true because an if-then rule applies  
with substitution (?X1<sub>UID530</sub>, ?X2<sub>UID535</sub>, ?X3<sub>Report</sub>)  
since 1.2.7.2.4.1 Rule: (forall (?x1 ?x2 ?x3)  
(<= (ctrl-contact ?x1 ?x2 ?x3)  
(ctrl-contact-method-011 ?x1 ?x2 ?x3)))  
and 1.2.12.1 Inferred: (ctrl-contact-method-011 UID530 UID535 Report)

1.2.12.1 (ctrl-contact-method-011 UID530 UID535 Report)  
is true because an if-then rule applies  
with substitution (?X1<sub>UID530</sub>, ?X2<sub>UID535</sub>, ?X3<sub>Report</sub>)  
since 1.2.7.2.4.2.1 Rule: (forall (?x1 ?x2 ?x3)  
(<= (hold-meeting ?x1 ?x2 ?x3)  
(hold-meeting-method-031 ?x1 ?x2 ?x3)))  
and 1.2.12.1.1 Inferred: (hold-meeting UID530 UID535 Report)

1.2.12.1.1 (hold-meeting UID530 UID535 Report)  
is true because an if-then rule applies  
with substitution (?X1<sub>UID530</sub>, ?X2<sub>UID535</sub>, ?X3<sub>Report</sub>)  
since 1.2.7.2.4.2.2.1 Rule: (forall (?x1 ?x2 ?x3)  
(<= (hold-meeting-method-031 ?x1 ?x2 ?x3)  
(record-meeting ?x1 ?x2 ?x3)))  
and 1.2.12.1.1.1 Inferred: (record-meeting UID530 UID535 Report)

1.2.12.1.1.1 (record-meeting UID530 UID535 Report)  
is true because an if-then rule applies  
with substitution (?from<sub>UID530</sub>, ?to<sub>UID535</sub>, ?content<sub>Report</sub>, ?time/3/11/2002, ?newid<sub>UID6057</sub>)  
since 1.2.7.2.4.2.2.2.1 Rule: (forall (?from ?to ?content)  
(<= (!record-meeting ?from ?to ?content)  
(exists (?time ?newid)  
(and (MeetingTakingPlace ?newid)  
(meetingTime ?newid ?time)  
(socialParticipants ?newid ?from)  
(socialParticipants ?newid ?to)  
(senderOfInfo ?newid ?from)  
(recipientOfInfo ?newid ?to)  
(itellocationaryForce ?newid ?content))))  
and 1.2.12.1.1.1.1 Fact: (socialParticipants UID6057 UID530)  
and 1.2.12.1.1.1.2 Fact: (MeetingTakingPlace UID6057)  
and 1.2.12.1.1.1.3 Fact: (socialParticipants UID6057 UID535)  
and 1.2.12.1.1.1.4 Fact: (senderOfInfo UID6057 UID530)  
and 1.2.12.1.1.1.5 Fact: (recipientOfInfo UID6057 UID535)  
and 1.2.12.1.1.1.6 Fact: (itellocationaryForce UID6057 Report)  
and 1.2.12.1.1.1.7 Fact: (dateOfEvent UID6057 3/11/2002)

1.2.13 (ctrl-contact UID535 UID524 Report)  
is true because an if-then rule applies  
with substitution (?X1<sub>UID535</sub>, ?X2<sub>UID524</sub>, ?X3<sub>Report</sub>)  
since 1.2.7.2.3.1 Rule: (forall (?x1 ?x2 ?x3)  
(<= (ctrl-contact ?x1 ?x2 ?x3)  
(ctrl-contact-method-009 ?x1 ?x2 ?x3)))  
and 1.2.13.1 Inferred: (ctrl-contact-method-009 UID535 UID524 Report)

1.2.13.1 (ctrl-contact-method-009 UID535 UID524 Report)  
is true because an if-then rule applies  
with substitution (?X1<sub>UID535</sub>, ?X2<sub>UID524</sub>, ?X3<sub>Report</sub>)  
since 1.2.7.2.3.2.1 Rule: (forall (?x1 ?x2 ?x3)  
(<= (ctrl-contact-method-003 ?x1 ?x2 ?x3)  
(contact-phone ?x1 ?x2 ?x3)))  
and 1.2.13.1.1 Inferred: (contact-phone UID535 UID524 Report)

1.2.13.1.1 (contact-phone UID535 UID524 Report)  
is true because an if-then rule applies  
with substitution (?p1<sub>UID535</sub>, ?p2<sub>UID524</sub>, ?content<sub>Report</sub>, ?pn1<sub>UID536</sub>, ?pn2<sub>UID524</sub>)  
since 1.2.7.2.3.2.2.1 Rule: (forall (?p1 ?p2 ?content)  
(<= (contact-phone ?p1 ?p2 ?content)  
(exists (?pn2 ?pn1)  
(contact-phone-method-030 ?pn1 ?pn2 ?content ?p1 ?p2)))  
and 1.2.13.1.1.1 Inferred: (record-phone-call UID536 UID525 Report UID535 UID524)

1.2.13.1.1.1 (record-phone-call UID536 UID525 Report UID535 UID524)  
is true because an if-then rule applies  
with substitution (?pn1<sub>UID536</sub>, ?pn2<sub>UID525</sub>, ?content<sub>Report</sub>, ?p1<sub>UID535</sub>, ?p2<sub>UID524</sub>)  
since 1.2.7.2.3.2.2.2.1 Rule: (forall (?pn1 ?pn2 ?content ?p1 ?p2)  
(<= (contact-phone-method-030 ?pn1 ?pn2 ?content ?p1 ?p2)  
(and (agentPhoneNumber ?pn1 ?pn2)  
(agentPhoneNumber ?p2 ?pn1)  
(record-phone-call ?pn1 ?pn2 ?content))))  
and 1.2.10.1.1.1.1 Fact: (agentPhoneNumber UID535 UID536)  
and 1.2.11.2.1.1.1.1 Fact: (agentPhoneNumber UID524 UID525)  
and 1.2.13.1.1.1.1 Inferred: (record-phone-call UID536 UID525 Report)

1.2.13.1.1.1.1 **(record-phone-call UID536 UID525 Report)**  
is true because an if-then rule applies  
with substitution (:from<sub>UID536</sub>, :to<sub>UID525</sub>, ?content/Report, :time/3/14/2002, ?newid/UID6058)  
since 1.2.7.2.3.2.2.4.1 **Rule:** (forall (?from ?to ?content)  
(= (record-phone-call ?from ?to ?content)  
(exists (?time ?newid)  
(and (MakingAPhoneCall ?newid)  
(datesOfEvent ?newid ?time)  
(callerNumber ?newid ?from)  
(receiverNumber ?newid ?to)  
(itellocationaryForce ?newid ?content))))))  
and 1.2.13.1.1.1.1.1 **Fact:** (receiverNumber UID6058 UID525)  
and 1.2.13.1.1.1.1.2 **Fact:** (callerNumber UID6058 UID525)  
and 1.2.13.1.1.1.1.3 **Fact:** (MakingAPhoneCall UID6058)  
and 1.2.13.1.1.1.1.4 **Fact:** (itellocationaryForce UID6058 Report)  
and 1.2.13.1.1.1.1.5 **Fact:** (dateOfEvent UID6058 3/14/2002)

1.2.14 **(record-ck3 UID524 UID15 UID535 UID530)**  
is true because an if-then rule applies  
with substitution (:customer/UID524, :victim/UID15, ?middleman1/UID535, ?middleman2/UID530,  
?newid/UID6060)  
since 1.2.14.1 **Rule:** (forall (?customer ?victim ?middleman1 ?middleman2)  
(= (record-ck3 ?customer ?victim ?middleman1 ?middleman2)  
(exists (?newid)  
(and (MurderForHire ?newid)  
(victim ?newid ?victim)  
(hitContractor ?newid ?customer)  
(mediators ?newid ?middleman1)  
(mediators ?newid ?middleman2))))))  
and 1.2.14.2 **Fact:** (hitContractor UID6060 UID524)  
and 1.2.14.3 **Fact:** (victim UID6060 UID15)  
and 1.2.14.4 **Fact:** (murderForHire UID6060)  
and 1.2.14.5 **Fact:** (mediators UID6060 UID535)  
and 1.2.14.6 **Fact:** (mediators UID6060 UID530)

1.2.16 **(first-degree-murder UID531 UID15 Volgograd)**  
is true to some part because an if-then rule applies  
with substitution (:X1/UID531, :X2/UID15, :X3/Volgograd)  
since 1.2.16.1 **Rule:** (forall (?X1 ?X2 ?X3)  
(= (first-degree-murder ?X1 ?X2 ?X3)  
(first-degree-murder-method-017 ?X1 ?X2 ?X3)))  
and 1.2.16.2 **Infowed:** (first-degree-murder-method-017 UID531 UID15 Volgograd)

1.2.16.2 **(first-degree-murder-method-017 UID531 UID15 Volgograd)**  
is true to some part because an if-then rule applies  
with substitution (:killer/UID531, :victim/UID15, :city/Volgograd)  
since 1.2.16.2.1 **Rule:** (forall (?killer ?victim ?city)  
(= (first-degree-murder-method-017 ?killer ?victim ?city)  
(and (observe ?killer ?victim)  
(killing ?killer ?victim knife ?city)  
(record-fd-murder ?killer ?victim))))))  
and 1.2.16.2.2 **Infowed:** (observe UID531 UID15)  
and 1.2.16.2.3 **Infowed:** (killing UID531 UID15 Knife Volgograd)  
and 1.2.16.2.4 **Infowed:** (record-fd-murder UID531 UID15)

1.2.16.2.2 **(observe UID531 UID15)**  
is true because an if-then rule applies  
with substitution (:X1/UID531, :X2/UID15)  
since 1.2.16.2.2.1 **Rule:** (forall (?X1 ?X2)  
(= (observe ?X1 ?X2)  
(observe-method-013 ?X1 ?X2)))  
and 1.2.16.2.2.2 **Infewed:** (observe-method-013 UID531 UID15)

1.2.16.2.2.2 **(observe-method-013 UID531 UID15)**  
is true because an if-then rule applies  
with substitution (:X1/UID531, :X2/UID15)  
since 1.2.16.2.2.2.1 **Rule:** (forall (?X1 ?X2)  
(= (observe-method-013 ?X1 ?X2)  
(record-observe ?X1 ?X2)))  
and 1.2.16.2.2.2.2 **Infewed:** (record-observe UID531 UID15)

1.2.16.2.2.2.2 **(record-observe UID531 UID15)**  
is true because an if-then rule applies  
with substitution (:killer/UID531, :victim/UID15, :time/2/27/2002, ?newid/UID6047)  
since 1.2.16.2.2.2.2.1 **Rule:** (forall (?killer ?victim)  
(= (record-observe ?killer ?victim)  
(exists (?time ?newid)  
(and (Observing ?newid)  
(datesOfEvent ?newid ?time)  
(objectsObserved ?newid ?victim)  
(perpetrator ?newid ?killer))))))  
and 1.2.16.2.2.2.2.2 **Fact:** (objectsObserved UID6047 UID15)  
and 1.2.16.2.2.2.2.3 **Fact:** (Observing UID6047)  
and 1.2.16.2.2.2.2.4 **Fact:** (perpetrator UID6047 UID531)  
and 1.2.16.2.2.2.2.5 **Fact:** (dateOfEvent UID6047 2/27/2002)

is true to some part because an if-then rule applies  
with substitution (:X1/UID531, :X2/UID15, :X3/Knife, :X4/Volgograd)  
since 1.2.16.2.3.1 **Rule:** (forall (?X1 ?X2 ?X3 ?X4)  
(= (killing ?X1 ?X2 ?X3 ?X4)  
(killing-method-032 ?X1 ?X2 ?X3 ?X4)))  
and 1.2.16.2.3.2 **Infewed:** (killing-method-032 UID531 UID15 Knife Volgograd)

1.2.16.2.3.2 **(killing-method-032 UID531 UID15 Knife Volgograd)**  
is true to some part because an if-then rule applies  
with substitution (:X1/UID531, :X2/UID15, :X3/Knife, :X4/Volgograd)  
since 1.2.16.2.3.2.1 **Rule:** (forall (?X1 ?X2 ?X3 ?X4)  
(= (killing-method-032 ?X1 ?X2 ?X3 ?X4)  
(record-killing ?X1 ?X2 ?X3 ?X4)))  
and 1.2.16.2.3.2.2 **Infewed:** (record-killing UID531 UID15 Knife Volgograd)

1.2.16.2.3.2.2 **(record-killing UID531 UID15 Knife Volgograd)**  
is true to some part because an if-then rule applies  
with substitution (:killer/UID531, :victim/UID15, :weapon/Knife, :city/Volgograd, :time/3/1/2002, ?newid/UID6048)  
since 1.2.16.2.3.2.2.1 **Rule:** (forall (?killer ?victim ?weapon ?city)  
(= (record-killing ?killer ?victim ?weapon ?city)  
(exists (?time ?newid)  
(and (Murder ?newid)  
(datesOfEvent ?newid ?victim)  
(perpetrator ?newid ?killer)  
(ctl-dead ?victim)  
(deviceTypeUsed ?newid ?weapon)  
(eventOccurredAt ?newid ?city))))))  
and 1.2.16.2.3.2.2.2 **Fact:** (ctl-dead UID15)  
and 1.2.16.2.3.2.2.3 **Unknown:** (deviceTypeUsed UID6048 Knife)  
and 1.2.16.2.3.2.2.4 **Fact:** (eventOccursAt UID6048 Volgograd)  
and 1.2.16.2.3.2.2.5 **Fact:** (perpetrator UID6048 UID531)  
and 1.2.16.2.3.2.2.6 **Fact:** (murder UID6048)  
and 1.2.16.2.3.2.2.7 **Fact:** (victim UID6048 UID15)  
and 1.2.16.2.3.2.2.8 **Fact:** (dateOfEvent UID6048 3/1/2002)

1.2.16.2.4 **(record-fd-murder UID531 UID15)**  
is true because an if-then rule applies  
with substitution (:killer/UID531, :victim/UID15, ?newid/UID6051)  
since 1.2.16.2.4.1 **Rule:** (forall (?killer ?victim)  
(= (record-fd-murder ?killer ?victim)  
(exists (?newid)  
(and (Murder ?newid)  
(victim ?newid ?victim)  
(perpetrator ?newid ?killer))))))  
and 1.2.16.2.4.2 **Fact:** (PremeditatedMurder UID6051)  
and 1.2.16.2.4.3 **Fact:** (Victim UID6051 UID15)  
and 1.2.16.2.4.4 **Fact:** (perpetrator UID6051 UID531)

1.2.17 **(transfer-money UID535 UID530 21875)**  
is true because an if-then rule applies  
with substitution (:payer/UID535, :payee/UID530, :amount/21875, :payee-account/UID529, :payer-account/UID537)  
since 1.2.17.1 **Rule:** (forall (?payer ?payee ?amount)  
(= (transfer-money ?payer ?payee ?amount)  
(exists (?payee-account ?payer-account)  
(transfer-money-method-012 ?payer-account ?payee-account ?amount)))  
and 1.2.17.2 **Infewed:** (transfer-money-method-012 UID537 UID529 21875 UID535 UID530)

1.2.17.2 **(transfer-money-method-012 UID537 UID529 21875 UID535 UID530)**  
is true because an if-then rule applies  
with substitution (:payer-account/UID537, :payee-account/UID529, :amount/21875, :payer/UID535, :payee/UID530)  
since 1.2.17.2.1 **Rule:** (forall (?payer-account ?payee-account ?amount ?payer ?payee)  
(= (transfer-money-method-012 ?payer-account ?payee-account ?amount)  
(and (accountHolder ?payer-account ?payer)  
(Record-payment ?payer-account ?payee-account ?amount)))  
and 1.2.17.2.2 **Fact:** (accountHolder UID537 UID535)  
and 1.2.17.2.3 **Fact:** (accountHolder UID529 UID530)  
and 1.2.17.2.4 **Infewed:** (Record-payment UID537 UID529 21875)

1.2.17.2.4 **(Record-payment UID537 UID529 21875)**  
is true because an if-then rule applies  
with substitution (:sender/UID537, :recipient/UID529, :amount/21875, :time/2/1/2002, ?newid/UID6037)  
since 1.2.17.2.4.1 **Rule:** (forall (?sender ?recipient ?amount)  
(= (Record-payment ?sender ?recipient ?amount)  
(exists (?time ?newid)  
(and (datesOfEvent ?newid ?time)  
(payer ?newid ?sender)  
(toPossessor ?newid ?recipient)  
(moneyTransferred ?newid ?amount))))))  
and 1.2.17.2.4.2 **Fact:** (payer UID6037 UID537)  
and 1.2.17.2.4.3 **Fact:** (paying UID6037)  
and 1.2.17.2.4.4 **Fact:** (toPossessor UID6037 UID529)  
and 1.2.17.2.4.5 **Fact:** (dateOfEvent UID6037 2/1/2002)  
and 1.2.17.2.4.6 **Fact:** (moneyTransferred UID6037 21875)

Failure

```

1.2.18 (transfer-money UID524 UID535 24709)
is true because an if then rule applies
with substitution (?payer UID524, ?payee UID535, ?amount 24709, ?payer-account UID537, ?payer-account UID539)
since 1.2.17.1 Rule: (forall (?payer ?payee ?amount)
  (<= (transfer-money ?payer ?payee ?amount)
    (exists (?payer-account ?payer-account)
      (transfer-money-method-012 ?payer-account ?payee-account ?amount ?)))
and 1.2.18.1 Infused: (transfer-money-method-012 UID539 UID537 24709 UID524 UID535)

1.2.18.1 (transfer-money-method-012 UID539 UID537 24709 UID524 UID535)
is true because an if then rule applies
with substitution (?payer-account UID539, ?payee-account UID537, ?amount 24709, ?payer UID524, ?payee UID535)
since 1.2.17.2.1 Rule: (forall (?payer-account ?payee-account ?amount ?payer ?payee)
  (<= (transfer-money-method-012 ?payer-account ?payee-account ?amount ?)
    (and (?accountHolder ?payer-account ?payer)
      (?accountHolder ?payee-account ?payee)
      (?record-payment ?payer-account ?payee-account ?amount))))
and 1.2.18.1.1 Fact: (accountHolder UID539 UID524)
and 1.2.17.2.2 Fact: (accountHolder UID537 UID535)
and 1.2.18.1.2 Infused: (record-payment UID539 UID537 24709)

1.2.18.1.2 (record-payment UID539 UID537 24709)
is true because an if then rule applies
with substitution (?sender UID539, ?recipient UID537, ?amount 24709, ?time 1/20/2002, ?newid UID6027)
since 1.2.17.2.4.1 Rule: (forall (?sender ?recipient ?amount)
  (<= (?record-payment ?sender ?recipient ?amount)
    (exists (?time ?newid)
      (and (?paying ?newid)
        (?dateOfEvent ?newid ?time)
        (?payer ?newid ?sender)
        (?toPossessor ?newid ?recipient)
        (?moneyTransferred ?newid ?amount))))))
and 1.2.18.1.2.1 Fact: (payer UID6027 UID539)
and 1.2.18.1.2.2 Fact: (Paying UID6027)
and 1.2.18.1.2.3 Fact: (toPossessor UID6027 UID537)
and 1.2.18.1.2.4 Fact: (dateOfEvent UID6027 1/20/2002)
and 1.2.18.1.2.5 Fact: (moneyTransferred UID6027 24709)

1.2.19 (transfer-money UID530 UID531 9252)
is true because an if then rule applies
with substitution (?payer UID530, ?payee UID531, ?amount 9252, ?payer-account UID554, ?payer-account UID529)
since 1.2.17.1 Rule: (forall (?payer ?payee ?amount)
  (<= (transfer-money ?payer ?payee ?amount)
    (exists (?payer-account ?payer-account)
      (transfer-money-method-012 ?payer-account ?payee-account ?amount ?)))
and 1.2.19.1 Infused: (transfer-money-method-012 UID529 UID554 9252 UID530 UID531)

1.2.19.1 (transfer-money-method-012 UID529 UID554 9252 UID530 UID531)
is true because an if then rule applies
with substitution (?payer-account UID529, ?payee-account UID554, ?amount 9252, ?payer UID530, ?payee UID531)
since 1.2.17.2.1 Rule: (forall (?payer-account ?payee-account ?amount ?payer ?payee)
  (<= (transfer-money-method-012 ?payer-account ?payee-account ?amount ?)
    (and (?accountHolder ?payer-account ?payer)
      (?accountHolder ?payee-account ?payee)
      (?record-payment ?payer-account ?payee-account ?amount))))
and 1.2.17.2.3 Fact: (accountHolder UID529 UID530)
and 1.2.19.1.1 Fact: (accountHolder UID554 UID531)
and 1.2.19.1.2 Infused: (record-payment UID529 UID554 9252)

1.2.19.1.2 (record-payment UID529 UID554 9252)
is true because an if then rule applies
with substitution (?sender UID529, ?recipient UID554, ?amount 9252, ?time 2/14/2002, ?newid UID6045)
since 1.2.17.2.4.1 Rule: (forall (?sender ?recipient ?amount)
  (<= (?record-payment ?sender ?recipient ?amount)
    (exists (?time ?newid)
      (and (?paying ?newid)
        (?dateOfEvent ?newid ?time)
        (?payer ?newid ?sender)
        (?toPossessor ?newid ?recipient)
        (?moneyTransferred ?newid ?amount))))))
and 1.2.19.1.2.1 Fact: (payer UID6045 UID529)
and 1.2.19.1.2.2 Fact: (Paying UID6045)
and 1.2.19.1.2.3 Fact: (toPossessor UID6045 UID554)
and 1.2.19.1.2.4 Fact: (dateOfEvent UID6045 2/14/2002)
and 1.2.19.1.2.5 Fact: (moneyTransferred UID6045 9252)

```

➤ Strict proof: ~1 second  
 ➤ Partial proof: ~2 minutes  
 ➤ Large space of potential partial proofs explored

# Conclusion

---

---

- PowerLoom is well-suited for the representation & reasoning tasks:
  - Full-function, robust and stable KR&R system
    - Expressive representation, reasoning, query language, storage, extensive API
    - Available in Java (useful for integration with Protégé)
  - Meta-representation & reasoning
  - Concepts, relations, contexts, rules, queries, etc. are all first-class citizens which can be represented and reasoned about
  - Explanation support for successful and failed reasoning
  - Sophisticated context & module system
    - Encapsulation, efficient inference, representation of assumptions
  - Sophisticated support for units & measures
  - Support for simple timepoint reasoning.