Knowledge Engineering

Semester 2, 2004-05

Michael Rovatsos
mrovatso@inf.ed.ac.uk

School of informatics

Lecture 9 – Automated Software Synthesis
11th February 2005

Where are we?

- New Block: Knowledge Synthesis
  - How to bring different sources of knowledge together?
  - In classical, "closed" view: combine rules, sub-routines, knowledge bases
  - In emerging "open" view: autonomous agents interacting in a common environment (Internet, mobile devices, e-commerce, etc.)
- Will devote most time on the latter
- Today: Closed view – Automated software synthesis

The Amphion System

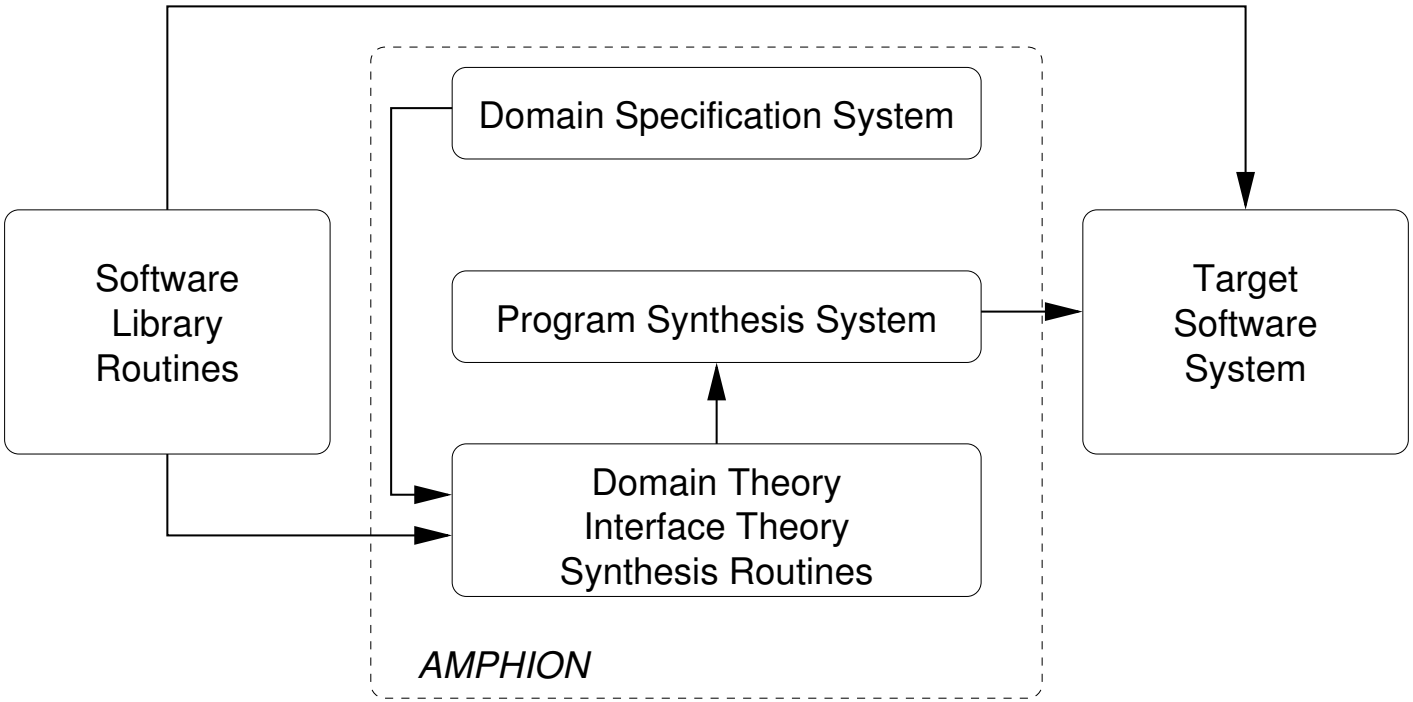
- System for automating software synthesis in the software engineering process
- Idea: Experts know a lot about domain, but little about programming ➡ provide them with high-level specification tool for automated software synthesis
- Also helps with software reuse problem
  - under-use of software libraries common
  - due to complexity of finding and combining existing routines
- Amphion = son of Zeus, charmed the stones lying around Thebes into position to form the city's walls (through the melodious sound of his magic lyre)

The Amphion System

The Amphion system comprises:

- A specification acquisition subsystem (generic):
  Used to build characterisation of domain
- A program synthesis subsystem (generic):
  Assembles programs from library routines
- A domain specific subsystem:
  Domain theory, interface routines, automation routines for synthesis system

The Amphion System



The Amphion System

- Used successfully to develop routines for controlling astronomical observations at NASA (using existing library of routines)
- Results: Programs can be assembled much faster, users comfortable with system after short tutorial
- Project homepage:
  `http://ase.arc.nasa.gov/docs/amphion.html`
- Approach based on **deductive synthesis**:
  Use the derivation for a query in a particular logic to derive a functional program that fits the specification

Specification & Derivation

- Assume a specification is given in terms of inputs, outputs, and constraints (like equations, functions, etc.) that have to be maintained
- First, system checks whether specification is satisfiable at abstract level
- Then, find a proof for the statement
  $\forall\, inputs\ \exists\, outputs\ spec(inputs, outputs)$
- A derivation of this proof gives a functional program F such that $\forall\, inputs\ spec(inputs, F(inputs))$
- Language-specific routines are only used after translating F to (say) an imperative language ➡ can be easily extended to different languages

Example

- Problem description: compute angle of sunlight at a point on a planet's surface
- Specify process by series of statments of the following form:

  Let *Solar-Incidence-Angle* be angle between rays *Surface-Normal* and *Ray-Intersection*
  Let *Surface-Normal* be ray normal to *Jupiter-Body* at the point *Boresight-Intersection*
  ...
- Can be seen as constraints in language of Euclidean geometry
- Use diagrammatic representation for manipulation by the user and synthesis

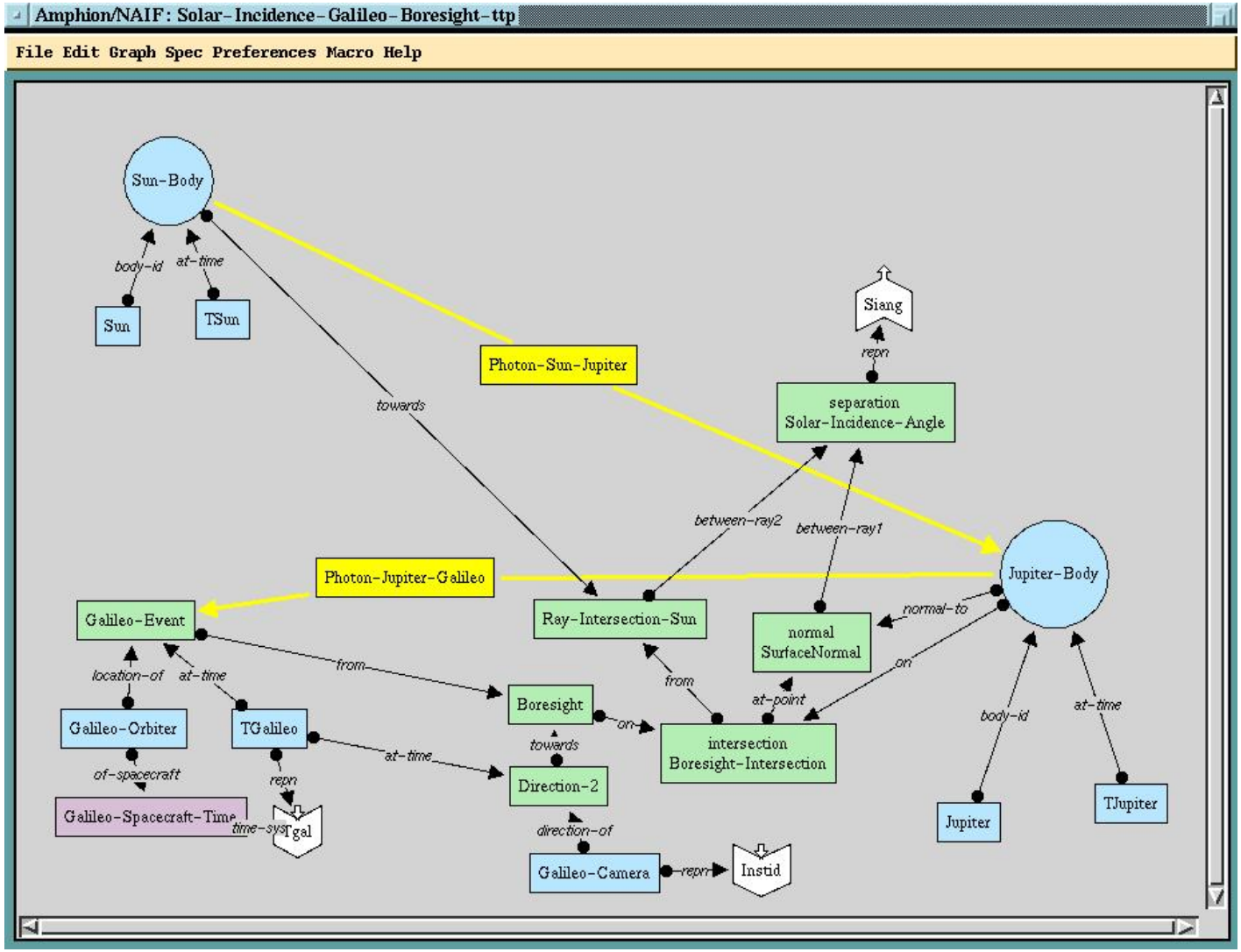
Domain Engineering

- Several aspects need to be specified:
  - Abstract theory of domain (e.g. geometry)
  - Concrete theory (describes functionality of subroutines)
  - Implementation relation btw. abstract and concrete theory
- In example: abstract theory consists of
  - types of objects (points, lines)
  - constructors (e.g. ray from point + direction)
  - geometric operations (e.g. intersection)

Further Issues

- Specification Acquisition
  - Through interaction of user with GUI interface
  - Top-down or bottom-up
  - Abstract check to preclude obvious mistakes
- Checking for well-formed input (wrt domain theory)
- Consistency checking: check for consistency between theory and specification query
- Completeness checking: is there enough information to compute output values?
- Graphical specification
  - System allows for adding/deleting/merging objects
  - Declaring inputs/outputs

Graphical User Interface



Summary

- Automated software synthesis with Amphion
- Illustrates combination of KBS with practical software engineering tasks
- Improves software reuse capabilities of libraries
- Graphical representations highly useful
- Next time: **Autonomous agents and multiagent systems**