The PROSPER Toolkit
A Tutorial

Richard Boulton, Graham Collins,
Louise Dennis, Tom Melham

University of Glasgow

ESPRIT Project LTR 26241
Introduction
Formal Verification Tools

• Current situation:
  – not integrated
  – internally monolithic
  – externally user-centric
  – expert user interaction

• Vision:
  – integrated proof engine
  – component-based
  – API-driven
  – hidden or push-button
  – user-oriented guidance

Design tools with `proof-engine inside’
The PROSPER Project

- Open proof-tool architecture
  - component toolkit for custom proof-engines
  - hardware and software applications
  - natural user interfaces/interaction
Toy Example - Excel

• Linear arithmetic (HOL)
• SAT (Prover Technology)

true/false/?

‘EQL(A5,C12)’

?- ∀x y z. E[A5] = E[C12]

• Mix of:
  – Windows/Unix/COM
  – VBASIC/Python/ML/C
• Easy to build:
  – 2-300 lines new code
  – days, not weeks
The PROSPER Architecture

- Plug-in components:
  - BDDs, SAT procedure
  - SVC, SMV, ACL2
  - HOL libraries

- Role of ML and theorem prover:
  - orchestrate verifications (scripting language)
  - semantics for combining results (higher order logic)
  - co-operating deduction and FSM algorithms
A Typical PROSPER Solution

- Design Tool
- Proof Engine
- Plugin e.g. SMV
- Plugin e.g. ACL2
A Bit More Detail

ML/HOL98 \[\rightarrow\] expr \[\leftarrow\] term

C++/SVC

check_valid \[\rightarrow\] thm \[\leftarrow\]

PII
term \[\rightarrow\] expr
expr \[\leftarrow\] term

true/false/counterexample

• PROSPER Toolkit:
  – data transport
  – control/interrupt handling
  – languages: C, ML, Java,...

• Research issues:
  – system engineering
  – semantic coherence
  – debugging/counterexamples
The PROSPER Tool Kit

- The PROSPER Integration Interface (PII): Code for inter-component communication
- A Core Proof Engine which understands theorems and inference rules
- Tools for building custom proof engines with APIs as extensions to the Core Proof Engine
- A Communication Manager
The Core Proof Engine
HOL98

- A modular theorem prover
  - a simple core
  - can be extended by loading libraries of logical theories and proof procedures
- Implemented in Moscow ML (Standard ML)
- Can also be programmed in ML
- Used as the basis of PROSPER proof engines
The HOL Logic

• A simply-typed classical higher order logic
• Terms
  – variables, e.g. $x$
  – constants, e.g. T, F, 0, +
  – function applications, e.g. $f x$
  – lambda abstractions, e.g. $\lambda x. x + 1$
• All terms have a type
Custom Proof Engines

- Core of HOL98
  + HOL98 libraries
  + External plugins
  + New proof procedures in ML
  + Glue code in ML
  + PROSPER server-side support
- Client-side support used to incorporate plugins
The PROSPER
Integration Interface
Integration Interface

- A language-independent specification for communication between components
- For client and server components
- Implementations in ML, C, and Java (client)
  - in natural style for the language
- Supports common data types
  + logical terms, types, and theorems
Interface Data

- Booleans, integers, characters, strings
- Logical terms, types, and theorems
- Lists of interface data
- Option type: SOME interface data or NONE
- Pairs of interface data
- References to functions (allows composition)
Operations and Results

• Operations
  – Call (command, time_limit, interface data)
  – Interrupt
  – Quit

• Results
  – Succeeded (interface data)
  – Failed (interface data)
The PII Layers

Component

Application Support Layer

Communication Handling Layer

Translation Layer

Transport Layer
Server-Side Support

- Add operations to the API database
- Register component
- Accept a connection
- Process call (multiple times)
  … until client quits
- (Maybe) accept another connection
Client-Side Support

• Register component
• Connect to server
• Call server (multiple times)
• (Maybe) interrupt server
• Quit server
Client Operations in ML

register_component
   {component_name = “pe”};
val plugin = new_server “plugin”;
val result =
   call_server
       (plugin, “command”, 10, []);
Client Operations in C

PII_register_component ("client");
pe = PII_new_server ("pe");
result =
    PII_call_server
    (pe, "command", 10,
     PII_mk_list(NULL));
Client Operations in Java

```java
try {
    pii = new PII("client");
    server = new Server(pii, "pe");
} catch (ProsperException exc) { };
idata = new PII_List();
result =
    server.call("command", 10, idata);
```
Developer Aspects
Theorem Prover Aspect

Plugins

Core Proof Engine

Custom Proof Engine

ML

HOL Libraries
Application Aspect

Design Tool

Design Tool Programming language

API

Custom Proof Engine
Plugin Aspect

Verification Tool

Verification Tool Programming Language

Integration Interface Library

ML

Integration Interface Library
Case Study
Case Study: Microsoft Excel

• Wanted to see how the methodology worked within a system over which we had no control
• Problems: Excel in a Windows/COM world, PROSPER in a UNIX/Sockets world
• Proof engine based on linear arithmetic and propositional logic decision procedures
Case Study: System Structure

Windows

- Excel
- COM Server/PII Client
- Strings

Unix

- Proof Engine
- Prover Plugin
- Interface Data

Strings Interface Data
Case Study: Features

- Theorem Prover and Application Aspects
- Uses an existing plugin from Prover Technology
- Proof Engine uses the plugin, a linear arithmetic decision procedure library, and “glue code”
- Most work in the Application Aspect
Advanced Features
PROSPER Database

- Design Tool
- Proof Engine
- Database

Database API (read/write)
Database API (read only)
Database API (read only)

Plugin e.g. SMV
Plugin e.g. ACL2

Plugin e.g. SMV
Plugin e.g. ACL2
PROSPER Database

- Provides access to pre-existing HOL theory hierarchy
- Proof-engine operations to extend theory with automatic shadowing in database
- Expressive queries
- Avoids deadlock
Client Query
Dynamic Update

![Diagram showing Dynamic Update process with Client, PE, and DB]

May 2001

ESPRIT Project: LTR 26241
Possible Scenario

Client

DB

PE

Plugin

Plugin
The Communication Manager
The Communication Manager

- Physical connections with CM provide virtual connections between components
- Allows
  - interrupting (remotely)
  - redirection of stdout and stderr
  - ease of starting components
Interrupts

- Interrupt operation sent by application
- Intercepted by CM
- Interrupt signal sent using the operating system
Redirection of Output

• When starting up a component, the CM can “grab” its output streams and redirect them

• Possible targets for redirection
  – discard the output
  – stdout or stderr of the CM
  – multiplexed on an Internet socket

• Location of socket returned to client when it connects to a server