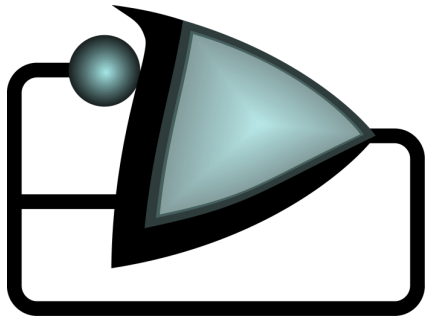
Logical Programming Environments

Jason Hickey, Richard Murray

John Hauser



What Formal Methods Offer

- Confidence:
  - **reliability:** we know what a system is supposed to do, and it does it
  - tools**:** specifications, code, and verification
- Automation:
  - code analysis, **synthesis**, and **optimization**
  - interactive design assistance
- High-confidence design requires systematic and structured approaches at all time scales (run time, design time)

Logical Programming Environments

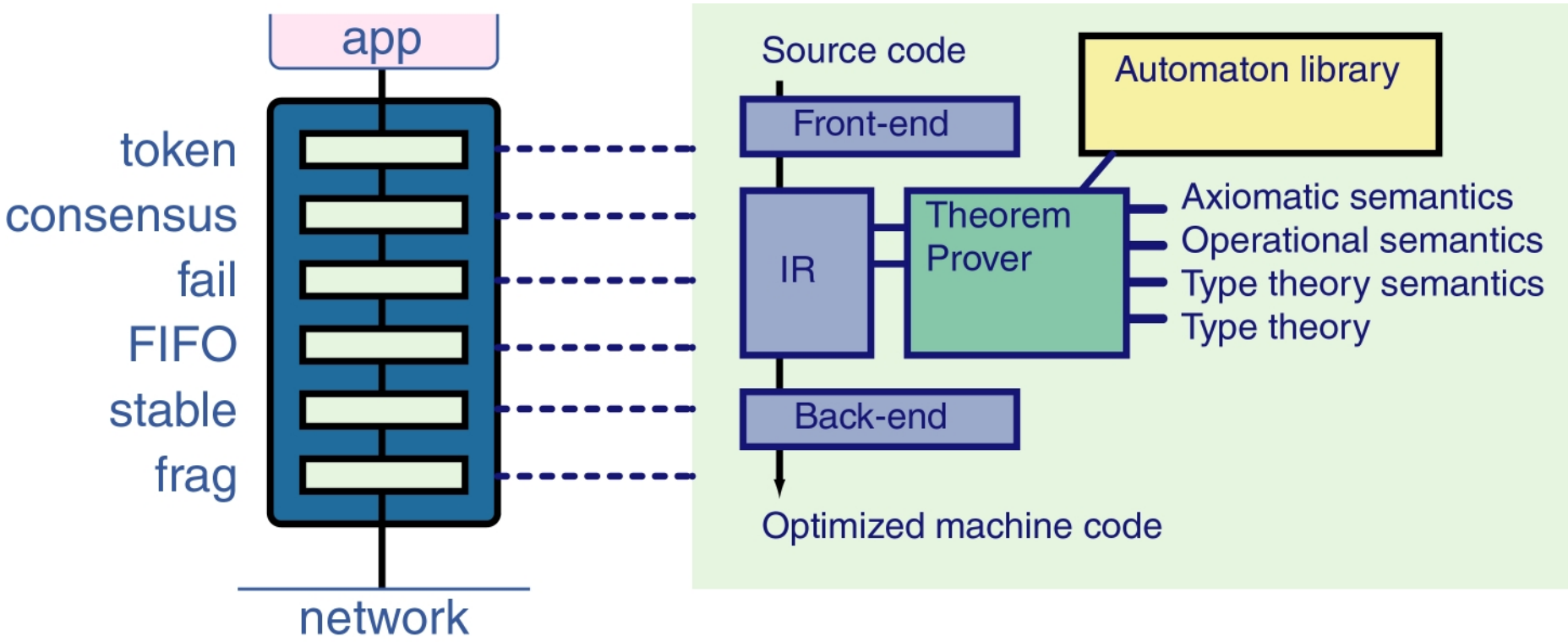
- A LPE provides an collaborative, interactive design environment
- An LPE includes:
  - A ***logical library*** where programs, proofs, and reasoning tools can be stored and shared in a collaborative development
  - A ***formal compiler*** that provides an open platform for producing executable code from programs, specifications, and proofs
  - An ***automated reasoning system*** that is used to develop formal proofs that programs meet their specification

An example

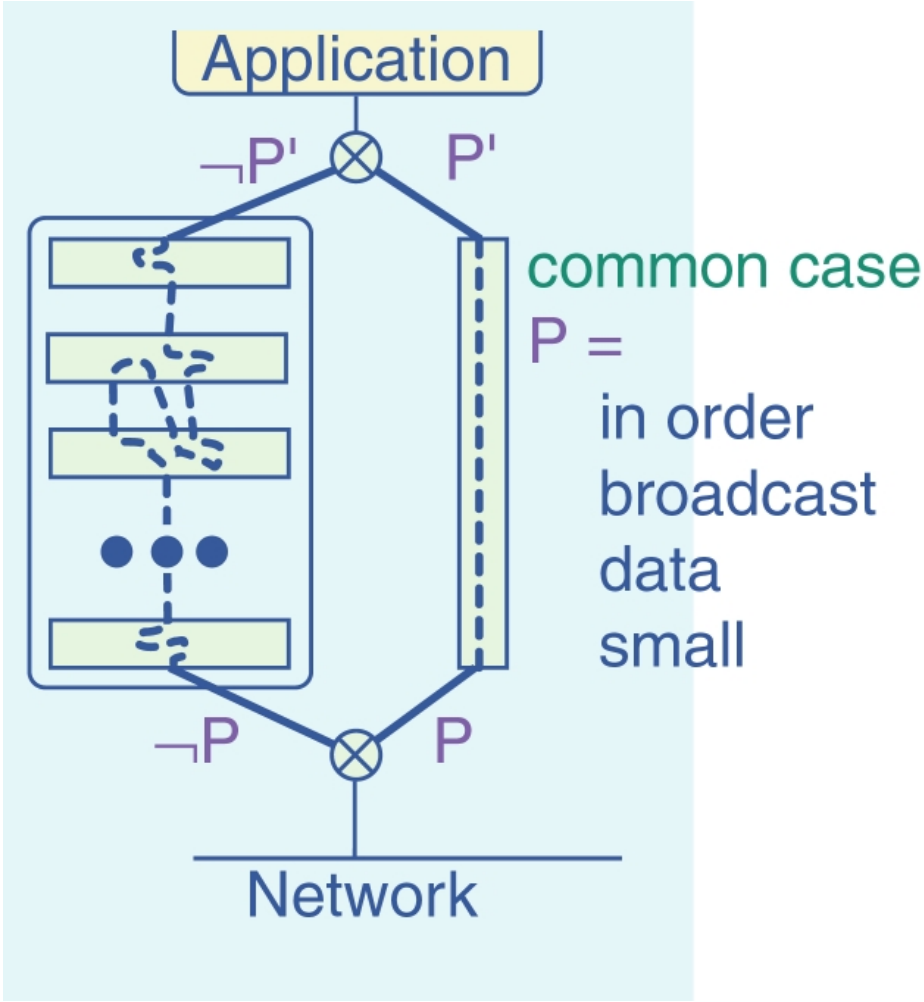
- Ensemble provides *group communication*
  - Like a multi-point version of TCP
  - Communication is reliable
- Used in NY, Swiss stock exchanges
- French air-traffic control
- Navy's AEGIS command, control

Formal tools

- Nuprl Logical Programming Environment
- All properties (and meta-proofs of algebra) are formal



Formal automation



- Protocols are pluggable components
- Protocol layers are in ML
- ~70 components, 1000s protocols
- About 30 layers in a protocol; roughly 300 lines of ML each
- Use refinement to verify/synthesize ML code

Applying the LPE to distributed control

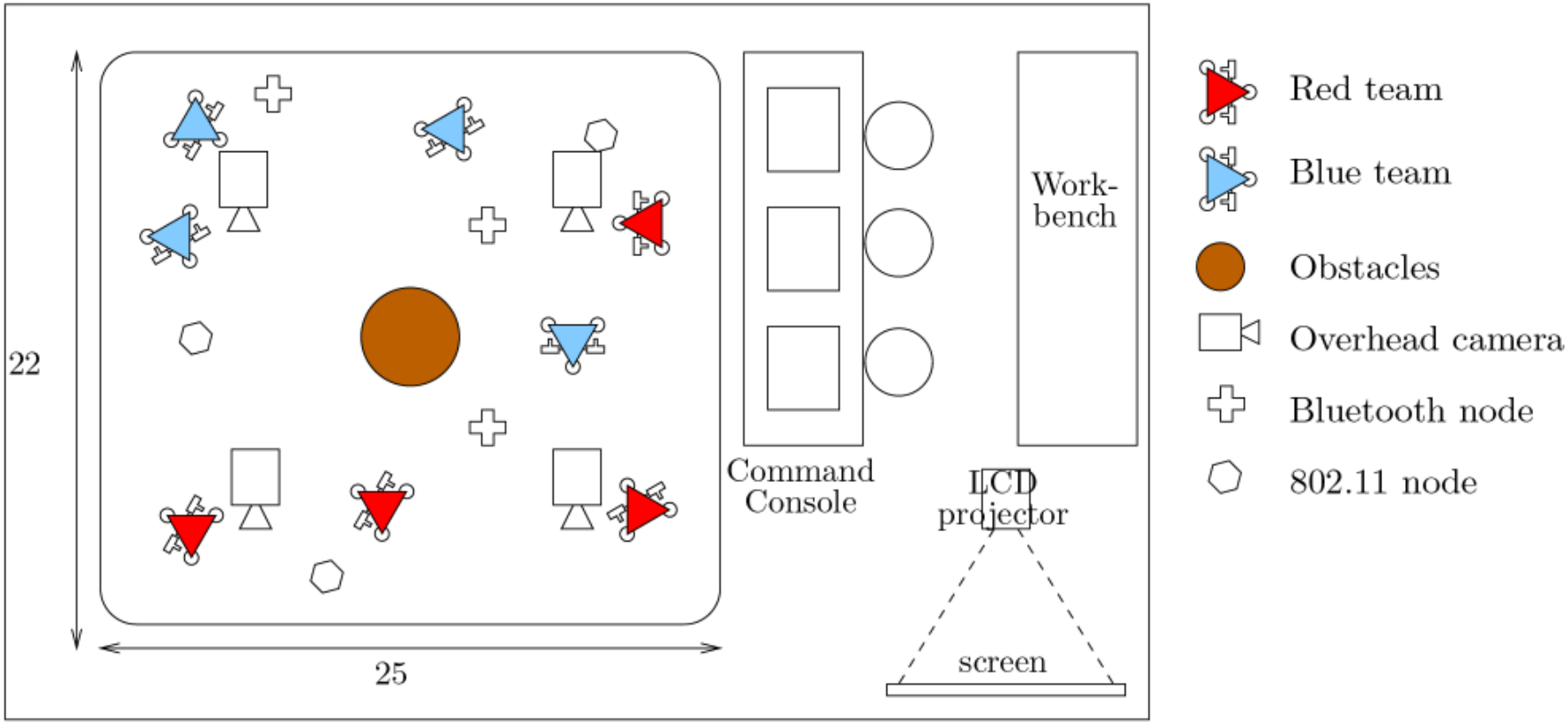
- Develop
  - A library of verified control components
  - A hierarchy of languages for cooperative control problems
  - A set of tools and heuristics for automated analysis and synthesis
- Design by successive *refinement*
  - Requirements propagate down
  - Assumption violations propagate upward (at design time and at run time)
  - *Interference* prevents straightforward composition

Multi-vehicle wireless testbed

- 8-10 vehicles, integrated computing and communications, including wireless Ethernet (802.11), and Bluetooth
- 2-4 fixed communication nodes, capable of broadcasting on multiple channels
- A set of overhead cameras that can be used to provide position information to the vehicles (perhaps simulating GPS)

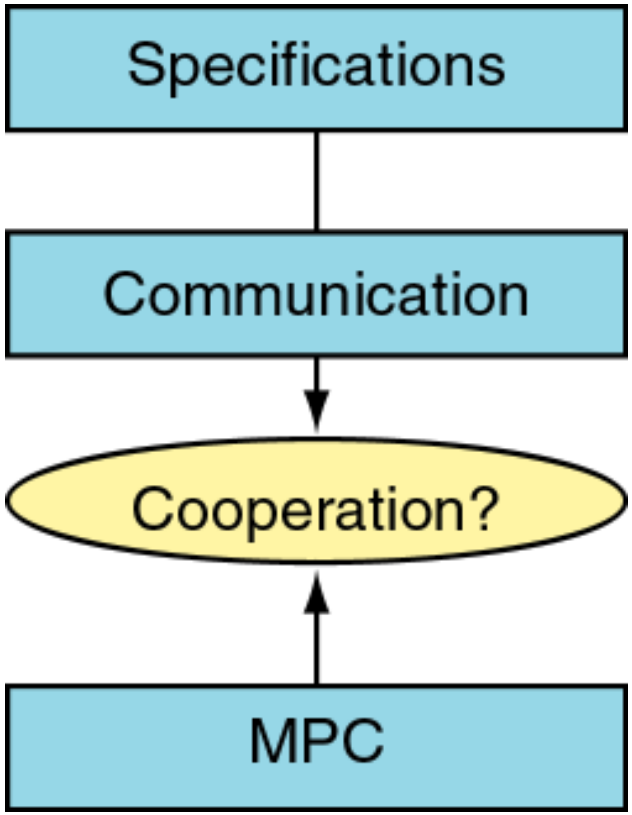
- A command console with computing and communication nodes

Multi-vehicle wireless testbed

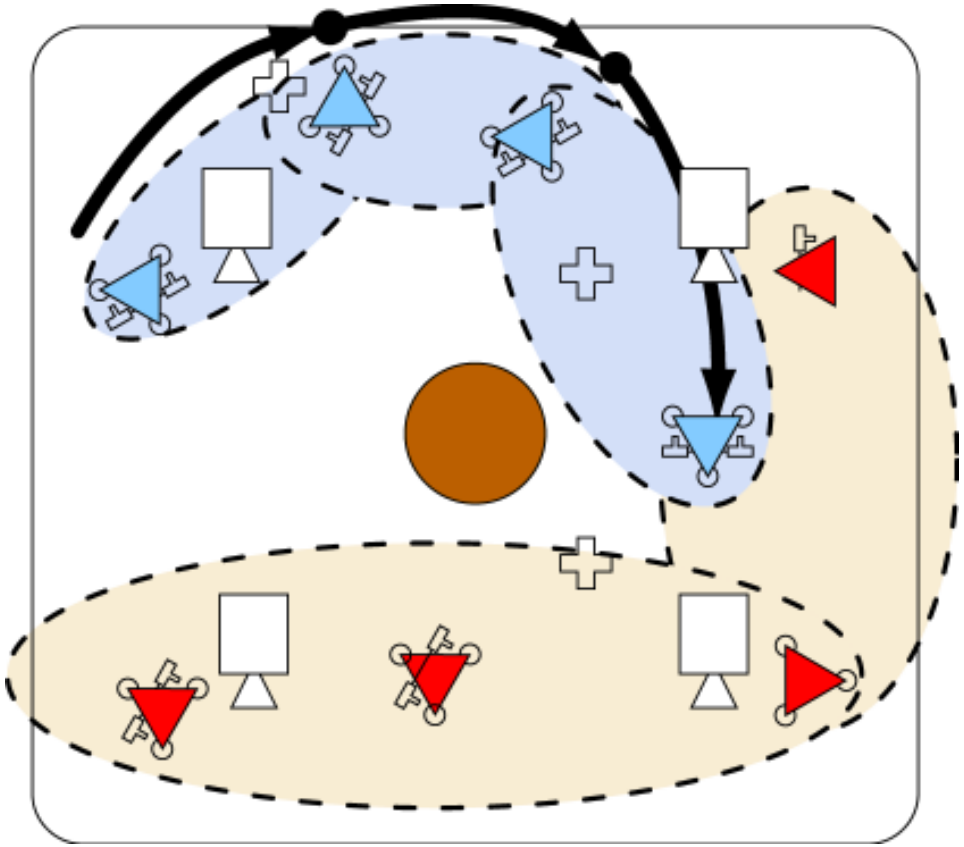


Current status

- Understand (to some extent)
  - high-level specifications
  - *asynchronous* communications
  - MPC
- Current focus
  - communication in rapidly-changing networks
  - design models for cooperative control



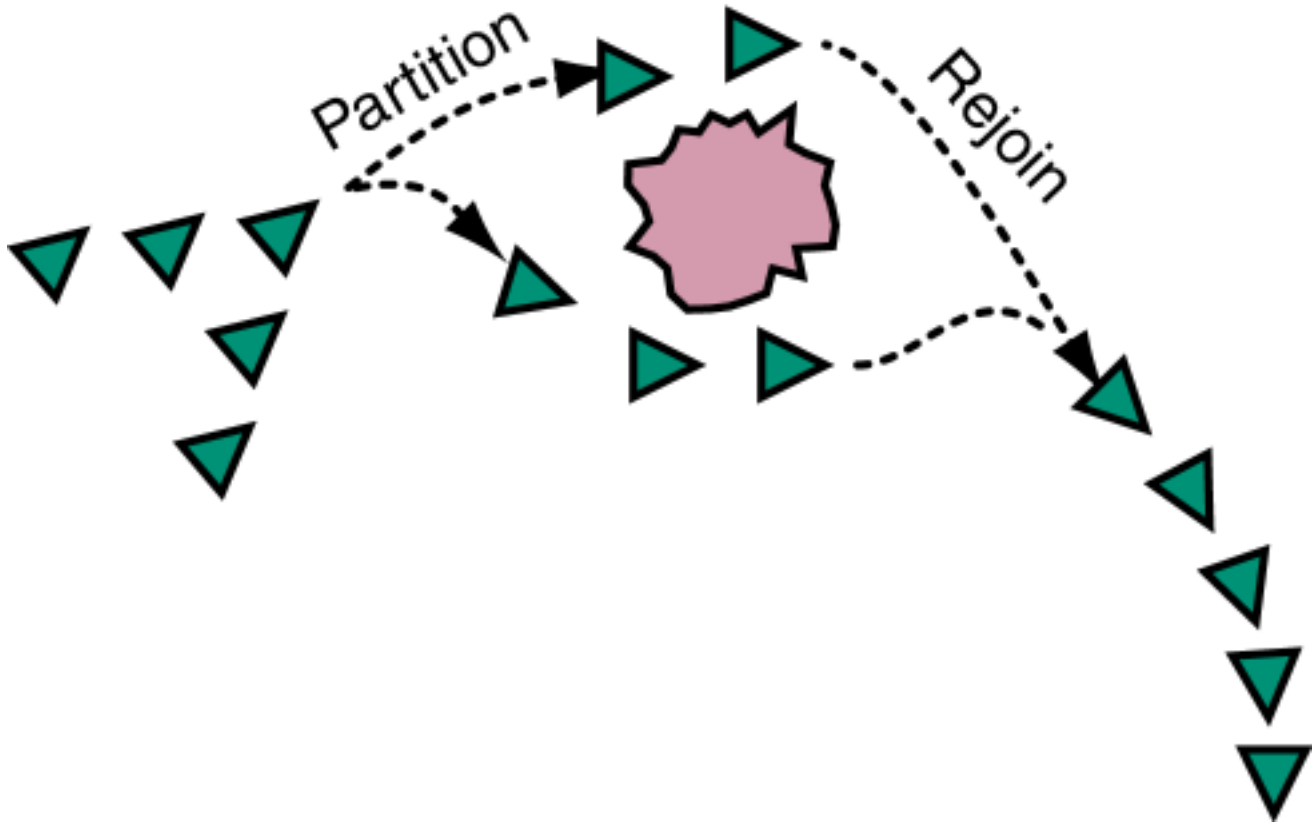
Multi-vehicle routing



- Network topology is rapidly changing
  - Consensus
  - Message routing
  - Real-time prioritized traffic
  - Make use of topology predictions

Problem formulation for UAV

- Formalize a rejoin



Top-level spec

- The *model* provides the basis for reasoning
- *Languages* provide the connection to syntax
- Top-level specification:

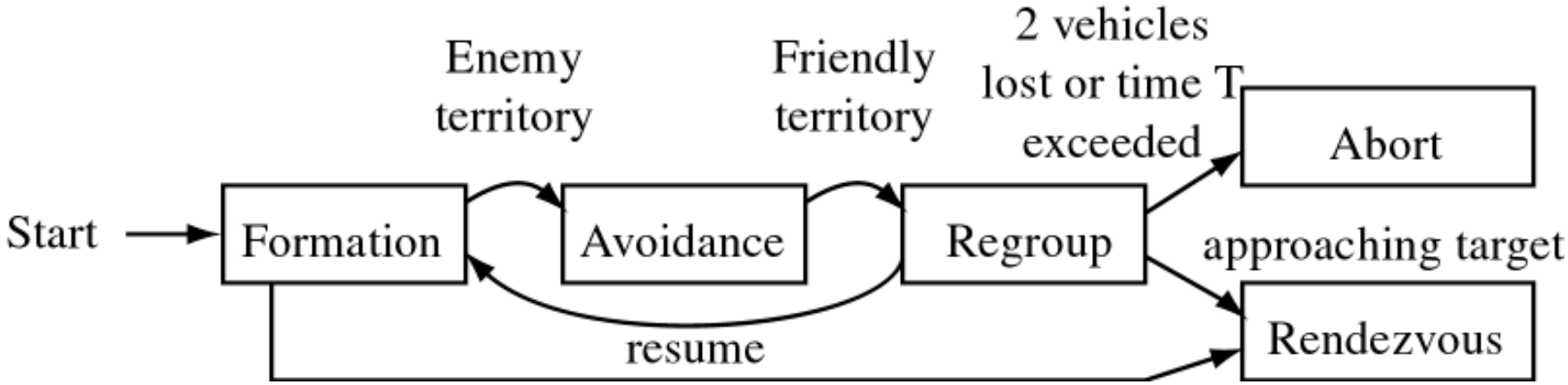
Mission Objective

Assumptions: $|operational_t(V)| \geq 4$

Goal: $\forall v \in V.\exists t \leq T.operational_t(v) \Rightarrow |v.pos_t - D| < \epsilon$

Second-level refinement

- Second-level: specify computation as a reactive state machine
- Verify that the decomposition satisfies the spec



Step refinement

- Each state is refined to an executable spec

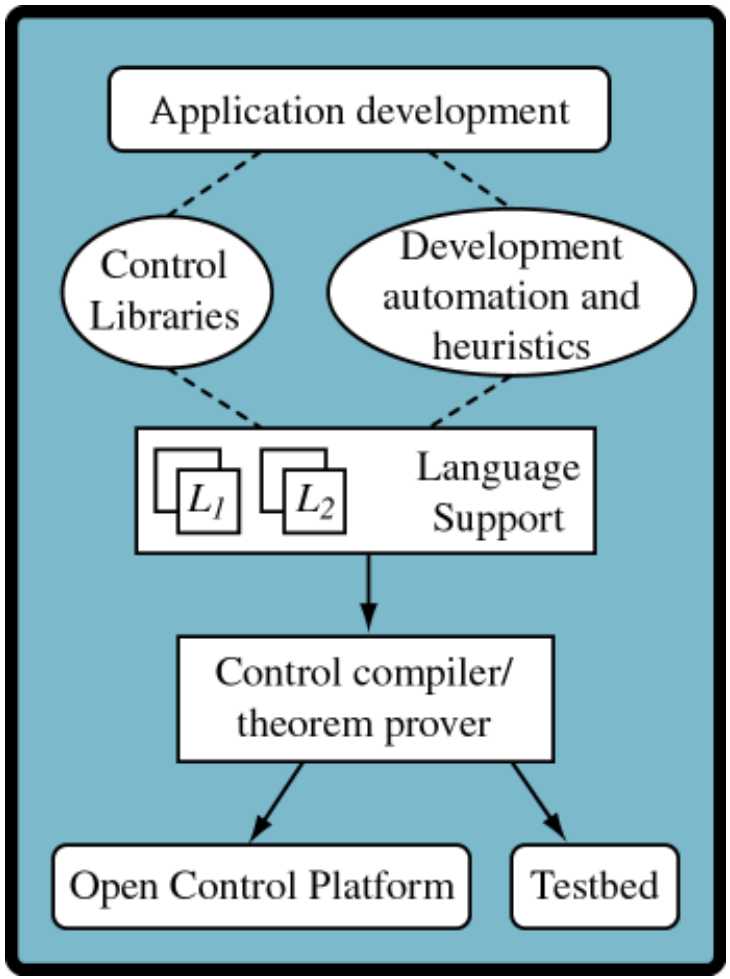
Choose destination vector

Assumptions: $bandwidth > bandwidth_{min}$
Goal: $Pre: Default \quad Eff: \mathbf{d}_v = projected\ formation\ point$
$Pre: Enemy\ detected \quad Eff: Abort$
$Pre: 2\ or\ more\ vehicles\ failed \quad Eff: Abort$

Move into formation

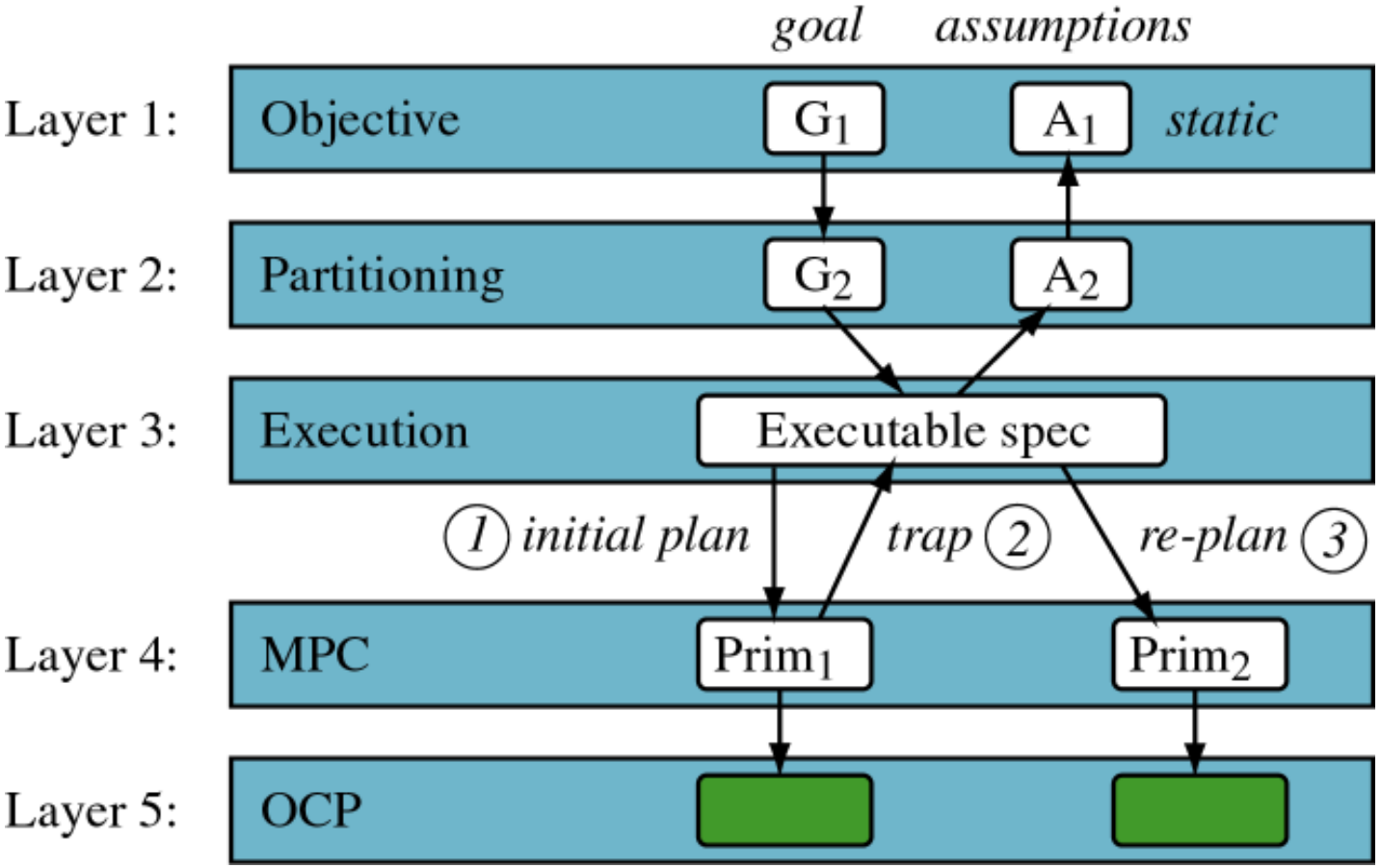
Assumptions: $bandwidth > bandwidth_{min}$
Goal: $Pre: Default \quad Eff: Continue\ to\ reform$
$Pre: Within\ tolerance \quad Eff: Resume\ formation$
$Pre: Enemy\ detected \quad Eff: Abort$

Logical Programming Environment



- The LPE is a framework for supporting formal design
  - *Type theory* is a common language for specification and synthesis
  - Enables *collaborative* development of verified control libraries and design automation tools
  - The *compiler* is an assistant, and the link to executable code

Design layers

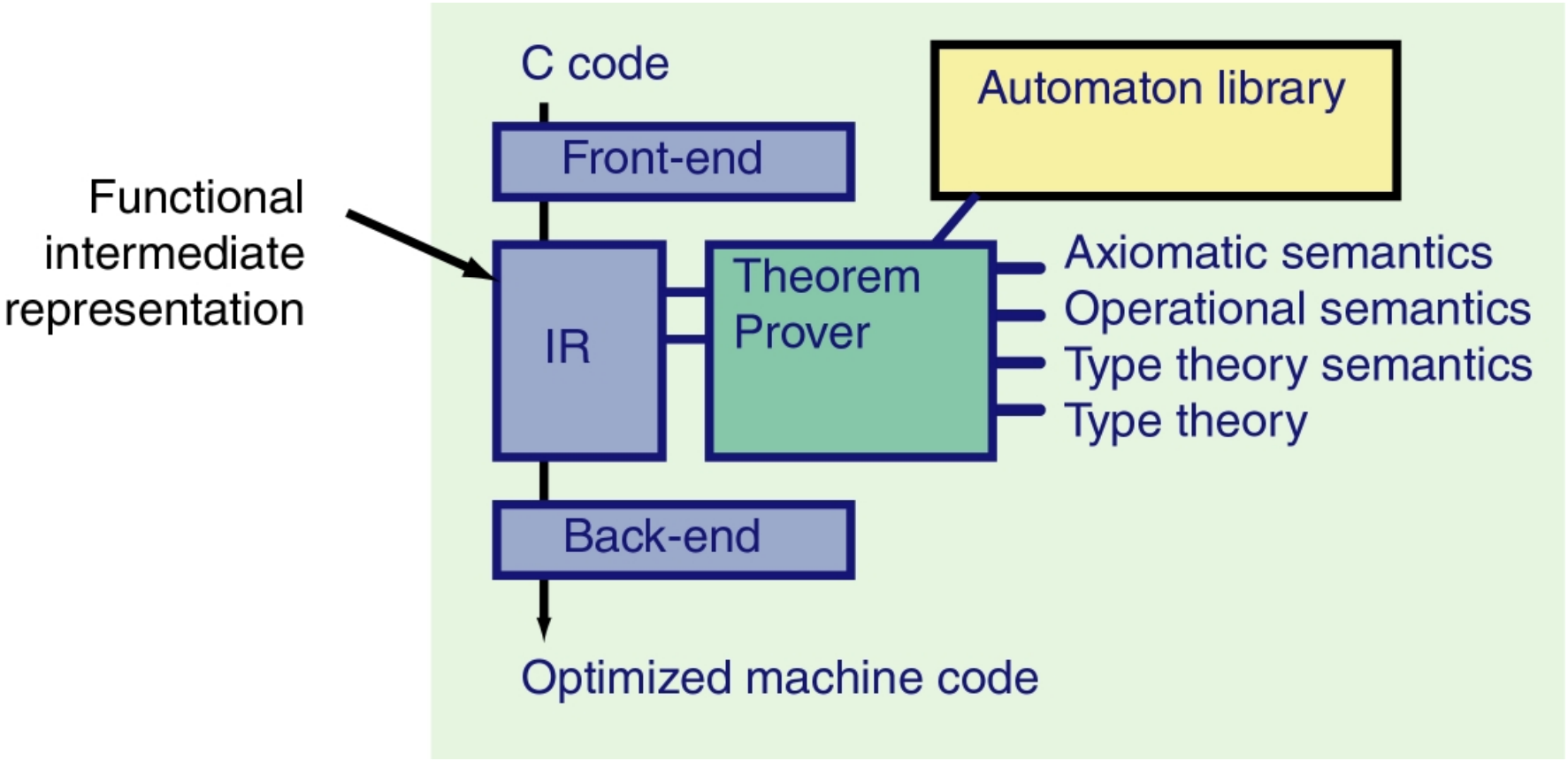


Migration path for legacy code: FC

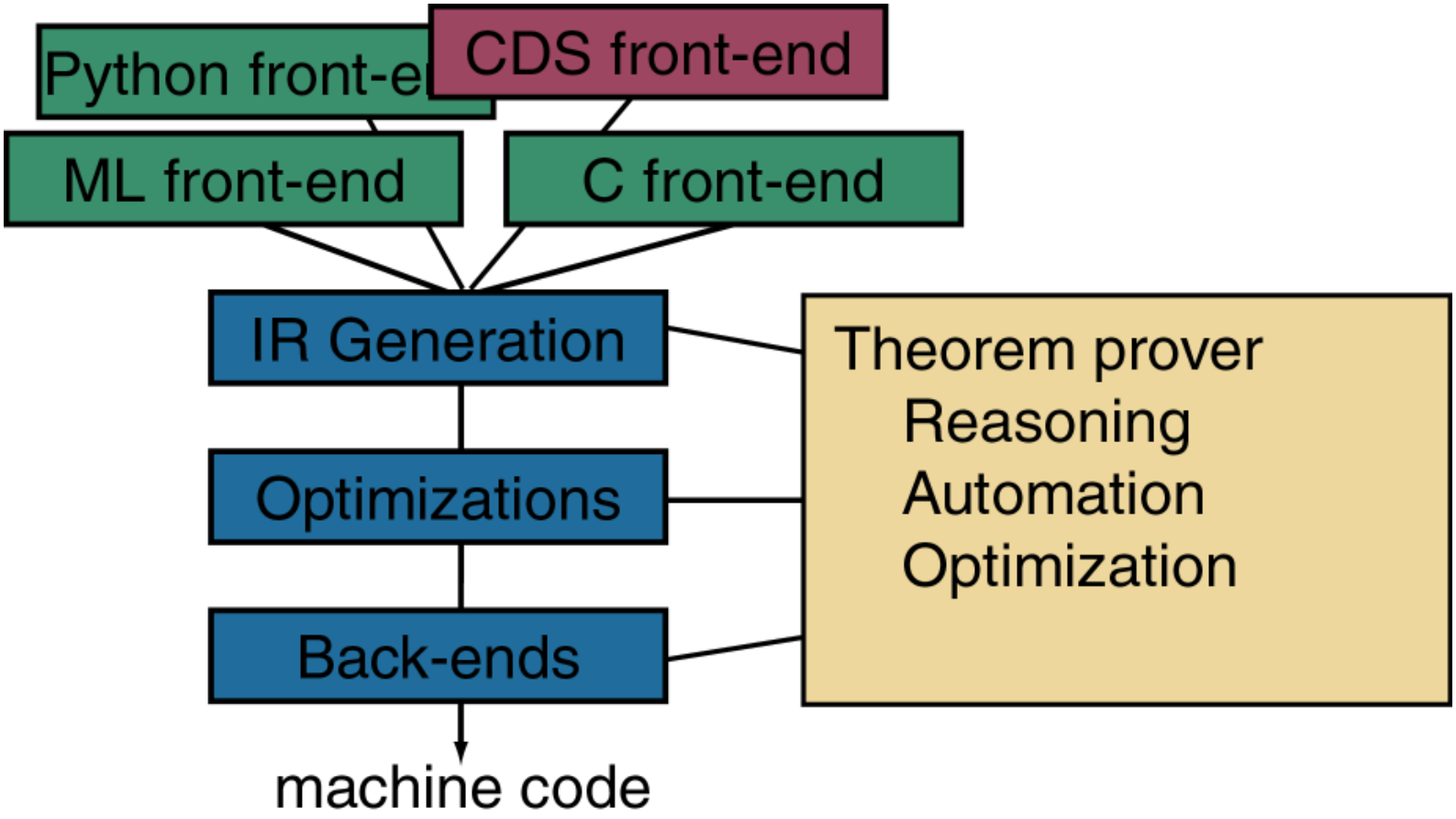
- Import C programs into a high-confidence, formal environment
- Allow *all* C programs
  - pointer arithmetic
  - arbitrary coercions
- Map to a **safe-** functional language
- Add: **transactions, migration**

$$
\begin{array}{lcl}
e & ::= & \textbf{let}\, v : t = a\, \textbf{in}\, e \\
 & | & \textbf{let}\, v = s\, \textbf{in}\, e \\
 & | & \textbf{let}\, v : t = unop\, a\, \textbf{in}\, e \\
 & | & \textbf{let}\, v : t = a\ binop\ a\, \textbf{in}\, e \\
 & | & \textbf{let type}\ typdefs\, \textbf{in}\, e \\
 & | & \textbf{let fun}\ fundefs\, \textbf{in}\, e \\
 & | & \textbf{let}\, v : t = f(a_1, \ldots, a_n)\, \textbf{in}\, e \\
 & | & \textbf{let closure}\, v : t = f(a_1, \ldots, a_n)\, \textbf{in}\, e \\
 & | & \textbf{let external}\, v = (f : ty)(a_1, \ldots, a_n)\, \textbf{in}\, e \\
 & | & f(a_1, \ldots, a_n) \\
 & | & \textbf{internal}\, f(a_1, \ldots, a_n) \\
 & | & \textbf{if}\, a_1\ relop\ a_2\, \textbf{then}\, e_1\, \textbf{else}\, e_2
\end{array}
$$

A formal C compiler



Multi-language environments



Summary

- LPE: leverage existing formal methods and tools for cooperative control problems
  - The goal is to provide a library of verified control primitives, and design automation procedures
- Migration path
  - The compiler provides the guide for migrating code