Security as a System-Level Constraint

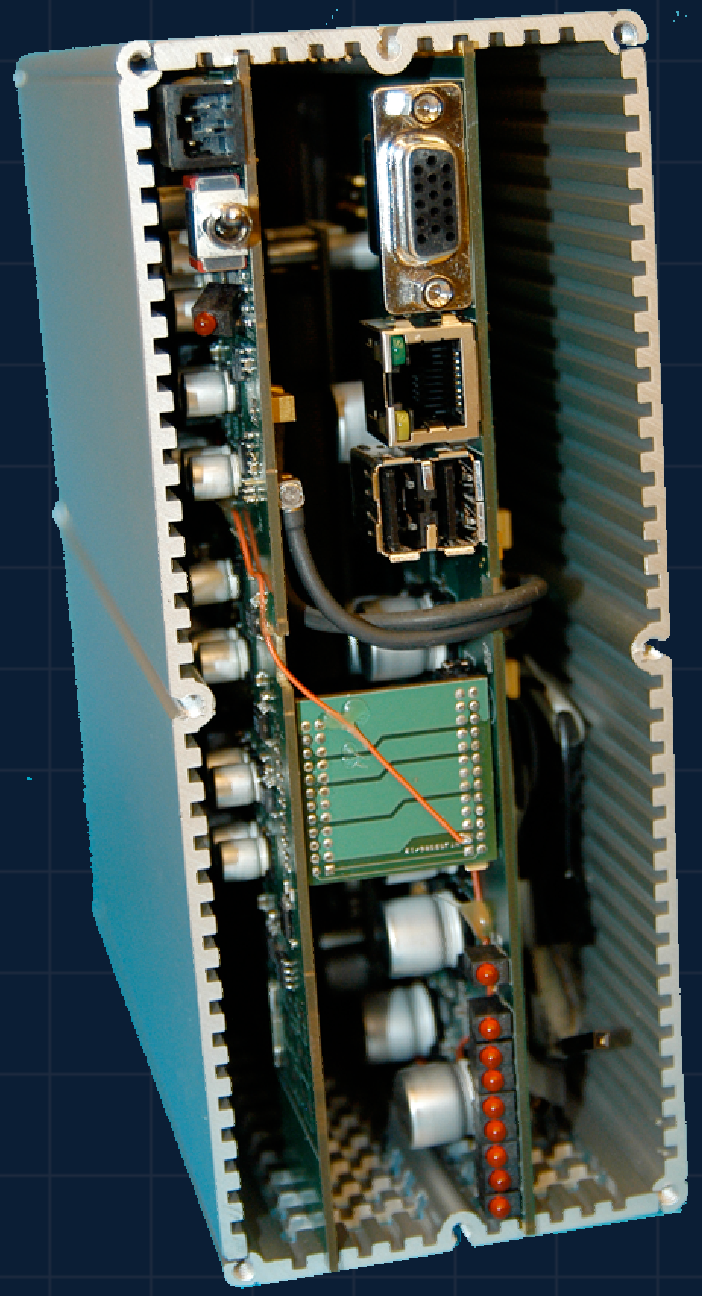
Dr. Perry Alexander
Information and Telecommunication Technology Center
Department of Electrical Engineering and Computer Science
The University of Kansas
alex@ittc.ku.edu

THE UNIVERSITY OF KANSAS

"The more I think about language, the more it amazes me that people ever understand each other"

- Kurt Gödel

Software Defined Cognitive Radios



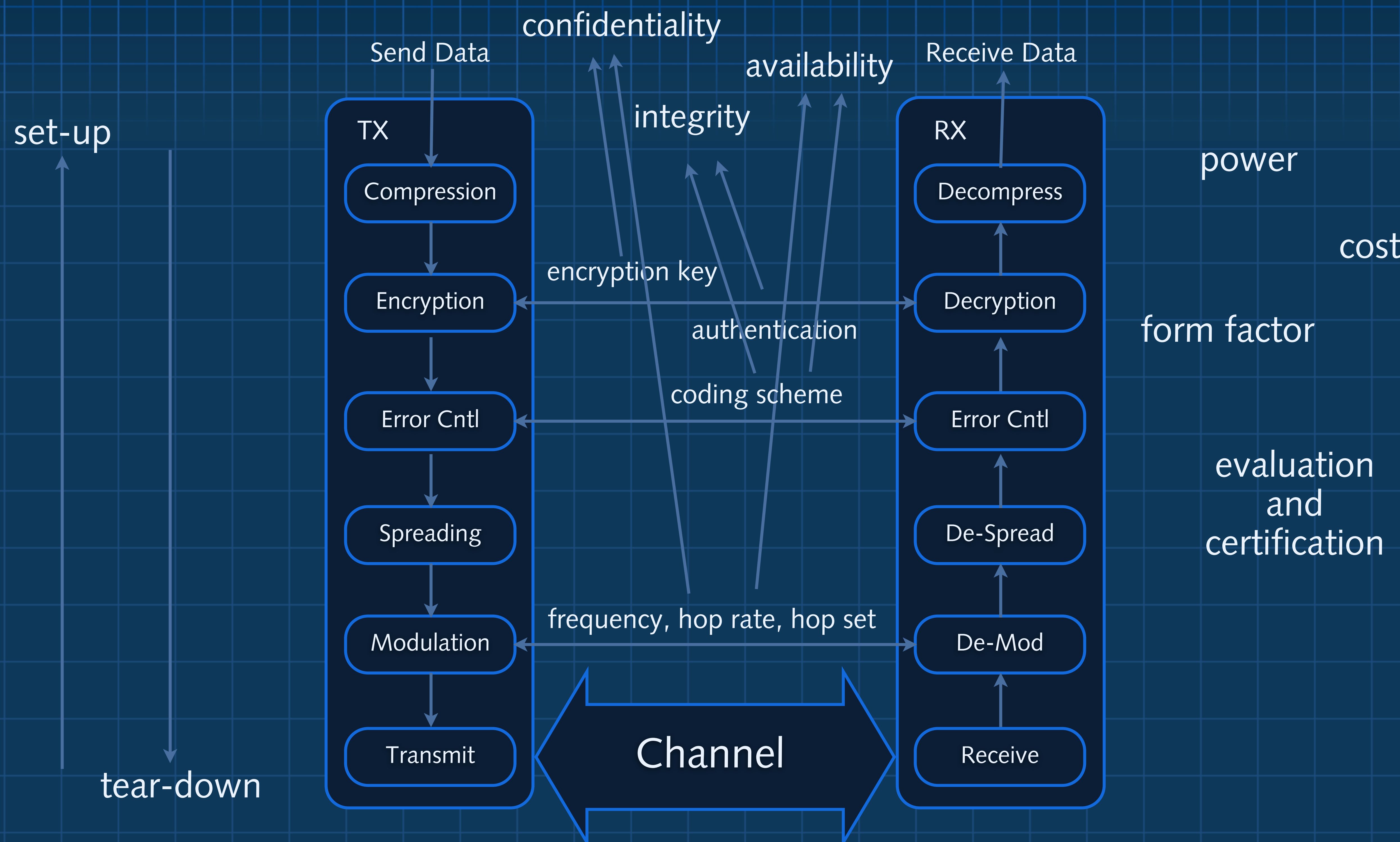
- Software Defined Radios
  - Design once, use many
  - Radios are commodity platforms
  - Waveforms are IP implemented on radios
- Cognitive Radios
  - Mission specific configuration
  - ElectroSpace resource management
  - Policy adherence
- Joint Tactical Radio System (JTRS)

THE UNIVERSITY OF KANSAS

SDR will implement...

- Soldier Radio Waveform (SRW)
- SINCGARS and Enhanced SINCGARS
- HAVE QUICK II
- UHF SATCOM
- Enhanced Position Locating Reporting System (EPLRS)
- Wideband Networking Waveform
- Link-4A, -11B, -16, and -22 tactical data links
- VHF-AM Air Traffic Control
- Anti-Jam Tactical Radio (NATO)
- Identification Friend or Foe (IFF)
- Cellular Telephone and PCS
- BOMAN UK Tri-Service HF, VHF, and UHF tactical comm
- and many, many more...

Waveform Architecture



THE UNIVERSITY OF KANSAS

SDR System-Level Modeling Issues

- Multiple heterogeneous perspectives
  - Information security
  - Analog and digital signal processing
  - Electromagnetic spectrum, energy, real-time, form factor, cost
- Heterogeneous domains
  - Continuous Time and Frequency domain
  - Digital signal processing
  - Temporal and/or Modal logic
- Multiple operating modes
  - Set-up
  - Operation
  - Tear-down
- System architecture
  - Component decomposition
  - Component integration

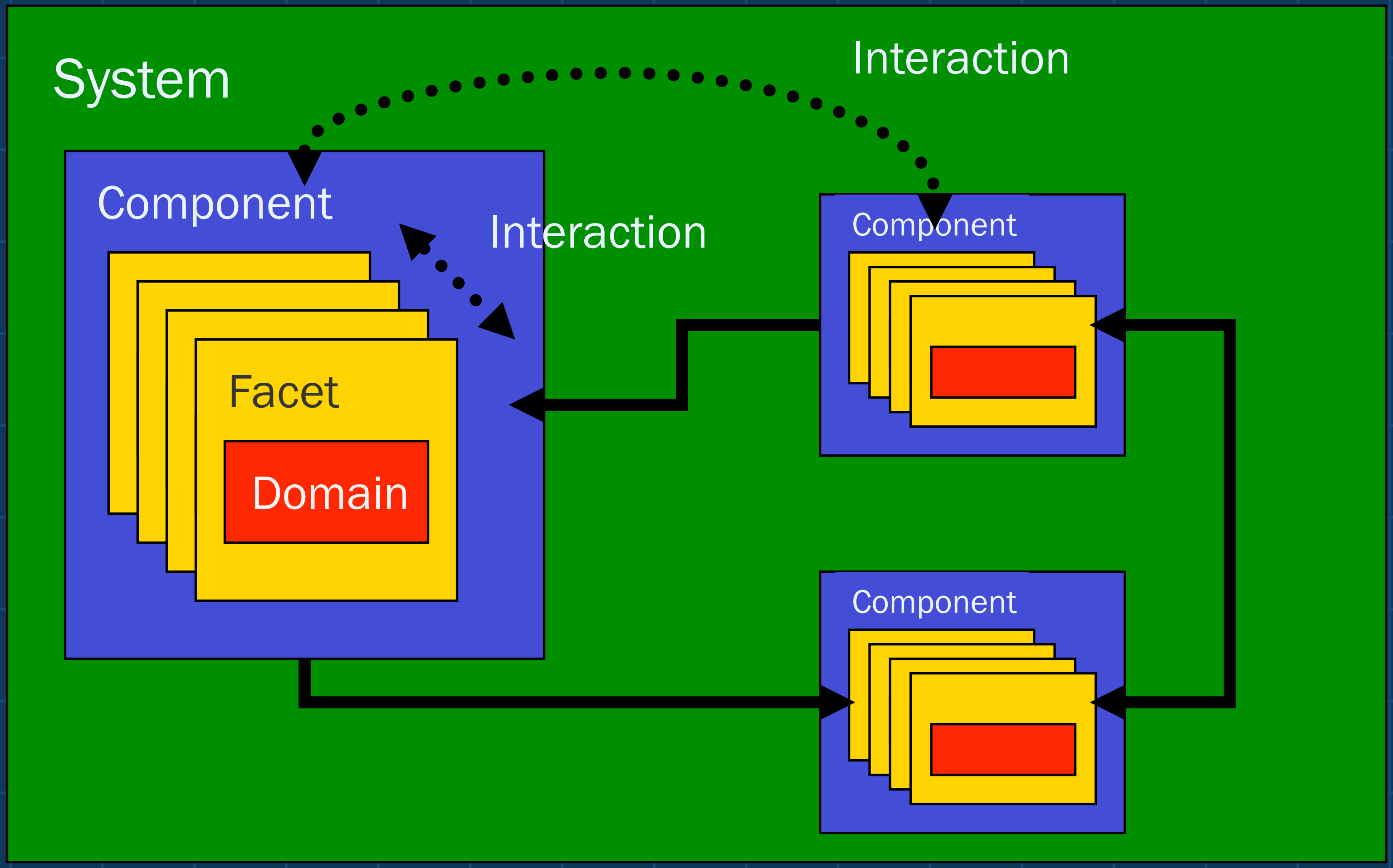
The Rosetta Language and Semantics

- Support for concurrent, system-level design
  - Facets and components for defining individual models
  - Domains for defining multiple computation models
  - Facet Algebra for defining model composition
  - Interactions for defining cross-domain implications
- Formal Semantics
  - Set theoretic, dependent type system
  - Coalgebraic semantics for environment specification
  - Algebraic semantics for local behavior specification
  - Category theoretic model composition operations
  - Reflection subsystem
- Heterogeneous, extensible domain system
  - Model-of-computation definitions
  - Lattice-based organization
  - Morphisms for domain transformation

What We Take from Rosetta

- Facets and Components
  - Define system and component models
  - Define system requirements
  - Record justifications
- Domain system
  - Define modeling semantics
  - Define specification transformations
  - Define specification interactions
- Facet algebra
  - Compose models
  - Define correctness conditions
- Analysis and synthesis capabilities

Anatomy of a Specification



THE UNIVERSITY OF KANSAS

Facet Definition Example

- A facet defines a coalgebraic system model
- Each facet extends a domain that defines a specification semantics
- Facet terms are boolean declarations or facet instances
- Facet parameters and variables define state and interface
- Functions encapsulate expressions
- All functions are pure with curried evaluation semantics

Facet Name — Parameter List — Domain

```
facet qam_mod_fun
  (i::input word(2); o::output real;
   en::input bit; f::design real) :: discrete_time is
begin
   o' = en*qamMod(f,t,i);
end facet qam_mod;
```

Terms

Function Name — Signature

```
qamMod(f,t::real,s::word(2))::real is
     amMod(kam,true,f,t,s(0))
   + amMod(kam,false,f,t,s(1));
```

Body

THE UNIVERSITY OF KANSAS

Component Definition Example

- A component defines a facet in context
- Component **definitions** play the same role as facet terms
- Component **assumptions** define usage assumptions
- Component **implications** define correctness conditions
- Justifications annotate assumptions and implications with evidence of truth
- assumptions AND definitions IMPLIES implications

```
component aes_mod_fcn
  (i::input blockType; o::output blockType;
   key::input keyType) :: discrete_time is
begin
  assumptions
     conf(key);
  end assumptions;
  definitions = aes_mod_req;
  end definitions;
  implications
    conf(o)
    and integ(o)
    and black(o)
        <== "rtc ase_mod.sld";
  end implications;
end component aes_mod_fcn;
```

Component Name → aes_mod_fcn

Parameter List → (i::input blockType; o::output blockType; key::input keyType)

Domain → discrete_time

Usage Assumptions → conf(key);

Requirements → aes_mod_req

Correctness Conditions → and integ(o)

Justification → <== "rtc ase_mod.sld";

Assembling Component Models

- Systems are defined by composing models
  - Components are instantiated
  - Composed in parallel
- System provides context for assumptions and implications

```
component waveform
    (din::input data; dout::output data;
     k::keyType)::radio is
  cd,ed,ecd,sd,md :: data;
begin
  assumptions
    conf(k);
  end assumptions;
  definitions
    c: compress(datain,cd);
    en: aes_mod_fcn(cd,ed,k);
    ec: errorCtl(ed,ecd);
    s: spread(ecd,sd);
    m: modulation(sd,md);
    t: transmit(md,dataout);
  end definitions;
  implications
    conf(dataout);
    black(dataout);
  end implications;
end facet waveform;
```

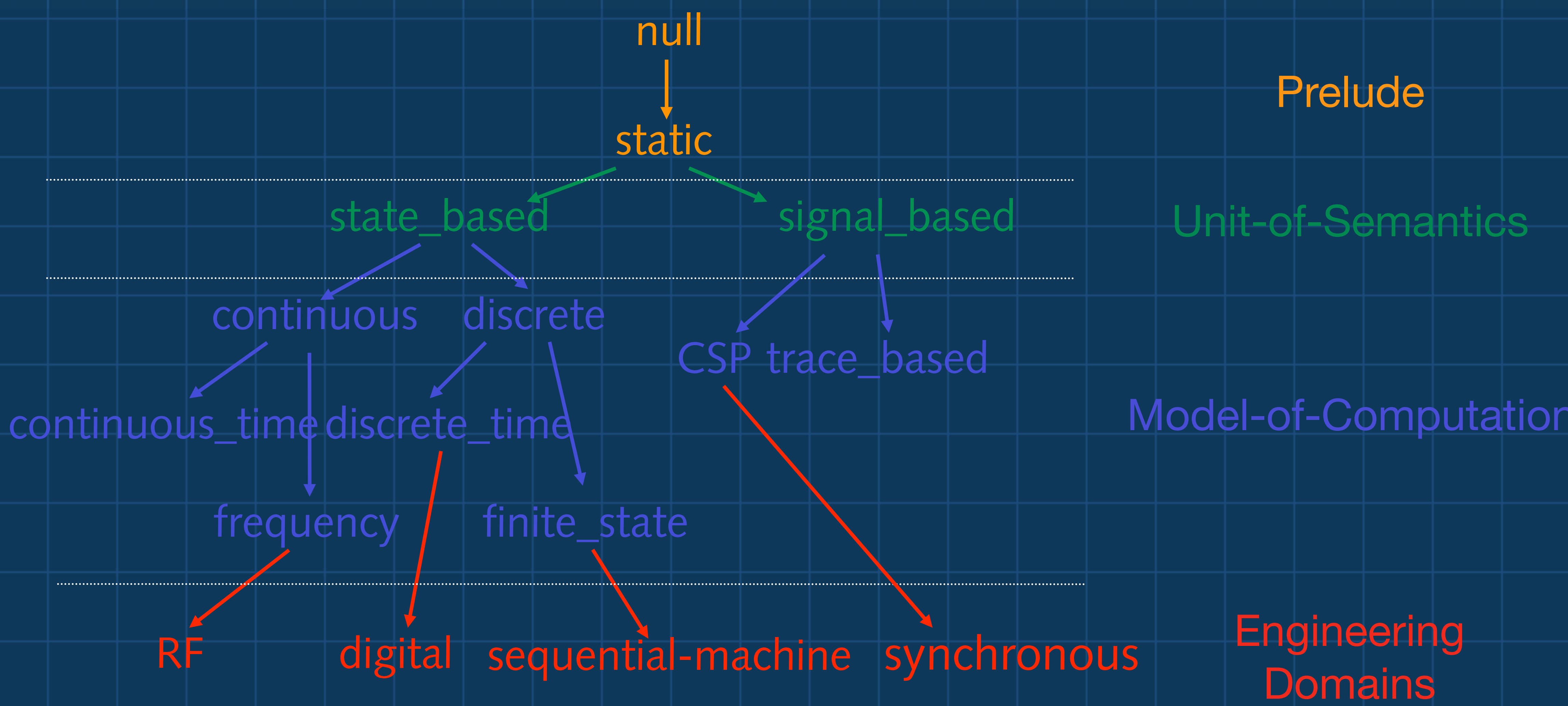
KU THE UNIVERSITY OF KANSAS

Modeling Domains

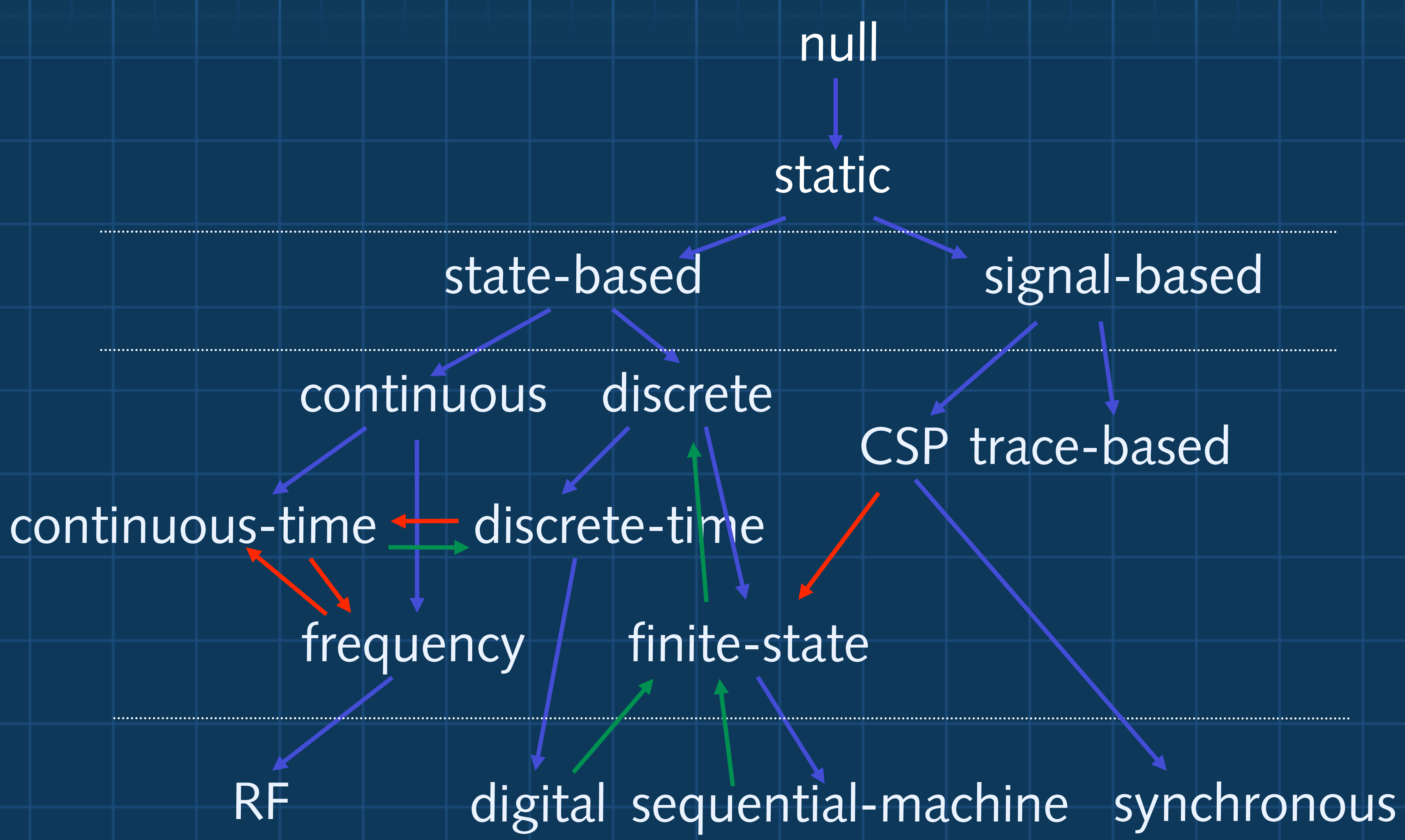
- Domains define vocabularies for specification
  - Semantic Units
  - Models of Computation
  - Engineering Vocabulary
- Domain definitions form a complete lattice
  - New domains extend existing domains resulting in homomorphisms
  - Functors define morphisms between domains
- Domain interactions model system-level properties
  - Define when information from one domain is linked to information in another
  - Models system impacts of decisions local to individual domains

THE UNIVERSITY OF KANSAS

The Domain Lattice



Functors



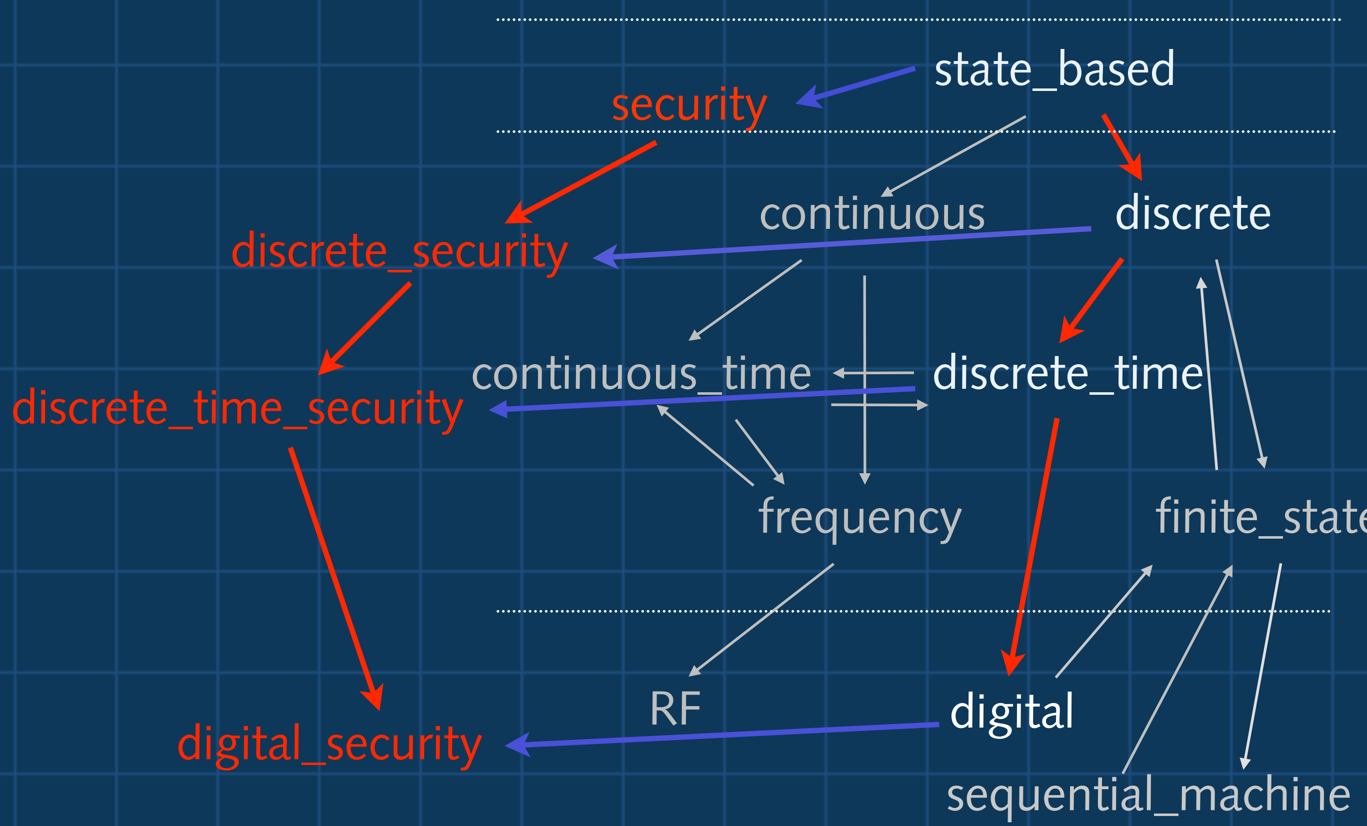
Extensions

Homomorphisms

General Functors
Abstraction, Concretization

Using Functors For Moving Models

- Domains are descriptors for categories of facets
  - The category consists of all extensions including bottom
  - The elements of the category ordered by homomorphism define a lattice
- A functor maps elements of one domain to another
  - Arrows to arrows and objects to objects
  - Functors operate on all elements of a domain



Using Functors for Refinement

```
domain security::state_based is
  riskType::type is posreal;
  p,nominal::riskType;
  activityType::subtype(real) is
     sel(x::real | x>=0.0 and x=<1.0);
  activity::activityType
begin
  p'=activity*nominal+latent;
end domain security;
```

Γ_{sb}

```
domain state_based::static is
  state_type::type; s::state_type;
  next(x::state_type)::state_type;
begin
end domain state_based;
```

$\Gamma \cdot \Gamma_{sb}$

$\Gamma_d \cdot \Gamma$

Γ

Γ

```
domain discrete_security :: discrete is
begin
  …
end domain discrete_security;
```

Γ_d

```
domain discrete::state_based is
begin
  state_type=natural;
end domain discrete;
```

Using the Galois Connection in the Lattice

- Establish soundness of abstractions and concretizations
  - Sound integration of new domains
  - Sound integration of synthesis and analysis tools
- (state_based,A,Γ,static) is a Galois connection
  - A is the abstraction function
  - Γ is the concretization function
  - We can calculate A when Γ is an extension
- No isomorphism unless $A \cdot \Gamma = id_{sb}$ and $\Gamma \cdot A = id_s$

```
domain state_based::static is
  state_type::type; s::state_type;
  next(x::state_type)::state_type;
begin
end domain state_based;
```

A ↑ ↓ Γ

```
domain security::state_based is
  riskType::type is posreal;
  p,nominal::riskType;
  activityType::subtype(real) is
     sel(x::real | x>=0.0 and x=<1.0);
  activity::activityType
begin
  p'=activity*nominal+latent;
end domain security;
```

Facet Composition

- Facet composition combines models to define systems
  - Vertical composition - Multiple views of a component
  - Horizontal composition - Multiple pieces of a component
- The facet algebra defines vertical composition
  - Facet algebra provides facet composition operations
  - Facet expressions define vertical composition
- Instantiating and renaming facets defines horizontal composition
  - Including instantiated facets as terms in models
  - Shared variables in interfaces facilitate communication

Facet Algebra Operations and Relations

Operation	Syntax
Product	$f_1 * f_2$ sharing {...}
Sum	$f_1 + f_2$ sharing {...}
Homomorphism	$f_1 => f_2$, $f_2 <= f_1$
Equivalence	$f_1 == f_2$
Functor	$F(f_1::D_1)::D_2$ is f_2
Instantiation	s: f(...);
Parallel Composition	s: f(...); t : f(...);

Composing Models

- Component aes_mod_fcn defines behavior and assumptions in operational mode
- Component aes_mod_boot defines behavior and assumptions during boot
- Component aes_mod disjointly composes models defining full operation

```
component interface aes_mod_fcn
  (i::input blockType; o::output blockType;
   key::input keyType) :: discrete_time
end component aes_mod;

component aes_mod_boot
  (i::input blockType; o::output blockType;
   key::input keyType) :: discrete_time
begin
  …
end component aes_mod_boot;

component aes_mod
  (i::input blockType; o::output blockType;
   key::input keyType) :: discrete_time is
       aes_mod_fcn(i,o,k)
       + aes_mod_boot(i,o,k);
       + aes_mod_teardown(i,o);
```

Composing Models

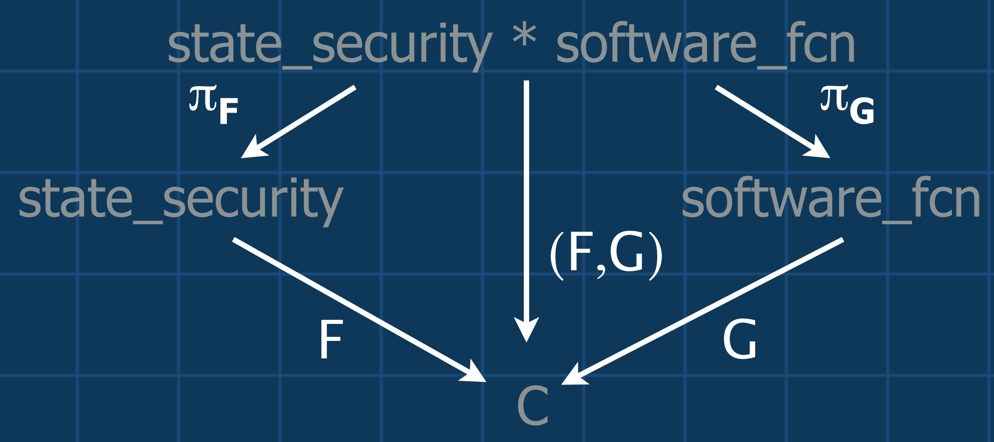
- Component aes_confidentiality defines an operational security policy
- Component aes_mod defines the operational system
- Component aes_mod_secure conjunctively composes models defining operation under security constraints

```
component aes_confidentiality
  (i::input blockType; o::output blockType;
   key::input keyType) :: discrete_time
end component aes_confidentiality;

component aes_mod
  (i::input blockType; o::output blockType;
   key::input keyType) :: discrete_time is
      aes_mod_fcn(i,o,k)
      + aes_mod_boot(i,o,k);
      + aes_mod_teardown(i,o,);

component aes_mod_secure
  (i::input blockType; o::output blockType;
   key::input keyType) :: discrete_time is
      aes_mod(i,o,k)
      * aes_confidentiality(i,o,k);
```

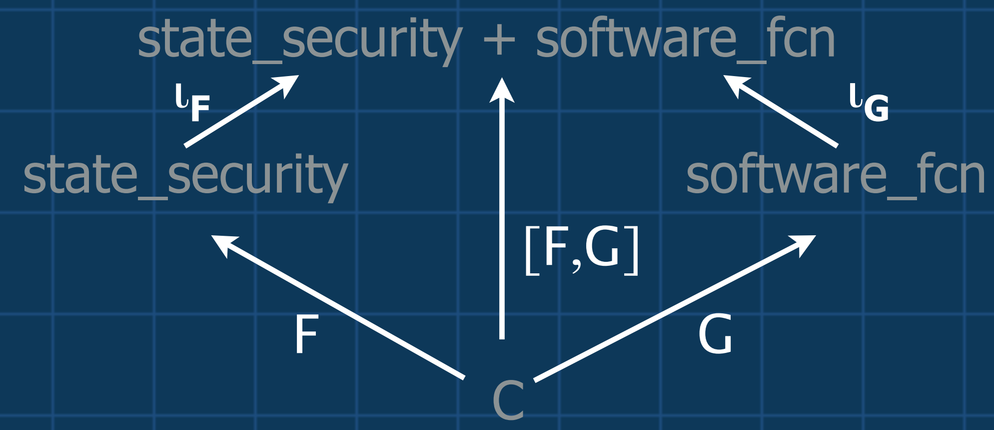
THE UNIVERSITY OF KANSAS

Facet Product and Sum

- A facet A*B is the product of facets
  - Product is parallel or conjunction
  - Result facet type is least fixed point



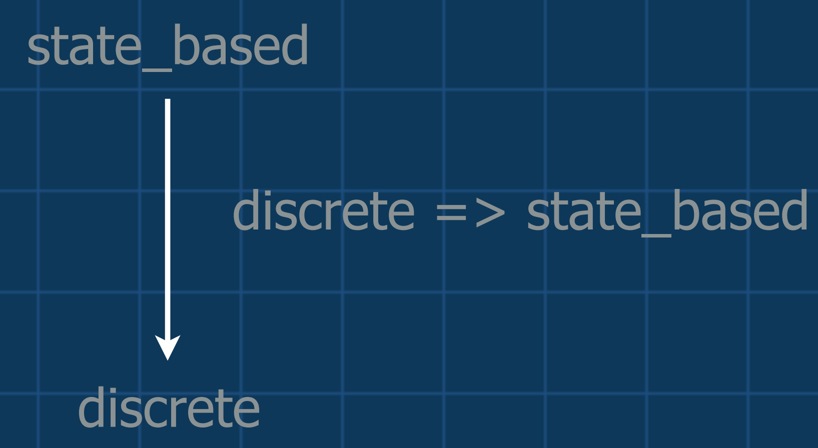
- A facet A+B is the sum of facets
  - Sum is alternative or disjunction
  - Result facet type is greatest fixed point



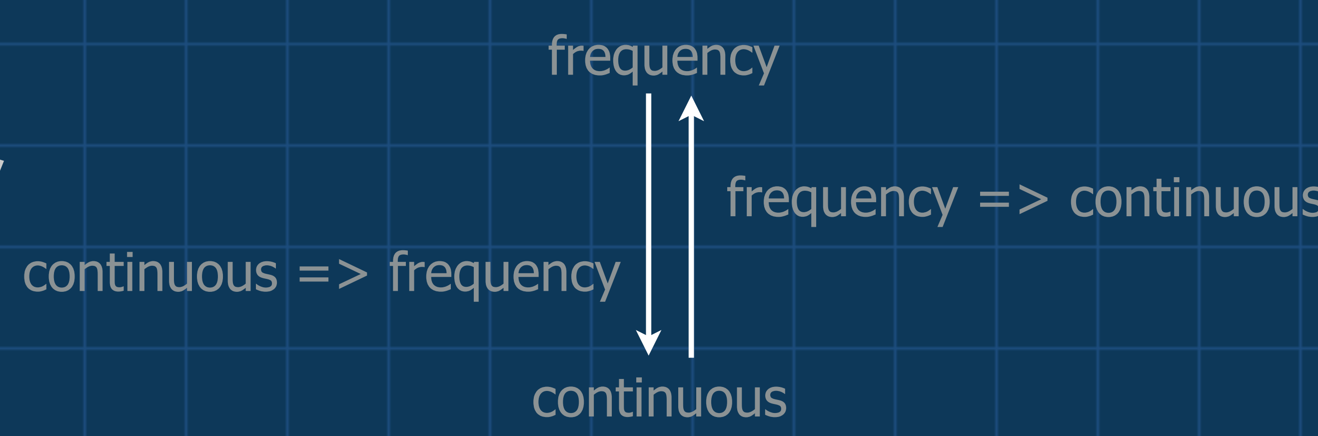
THE UNIVERSITY OF KANSAS

Homomorphism and Isomorphism

- Homomorphism defines the domain and facet lattices
  - Domains and facets are partially ordered by homomorphism
  - Top and bottom are null and bottom respectively

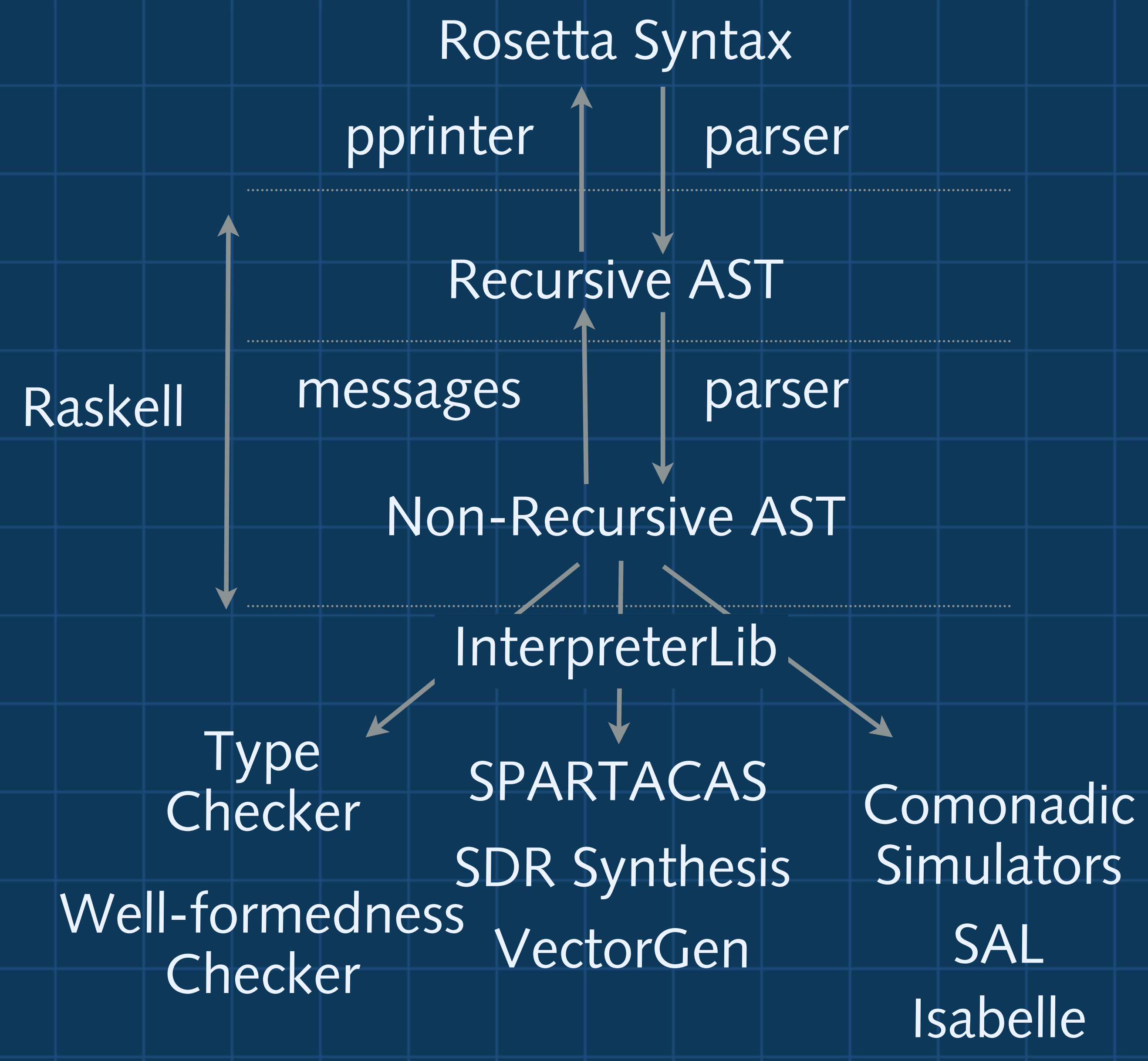


- Isomorphism is equivalence
  - Antisymmetric property of the homomorphism
  - Must be homomorphism, not simply functor existence



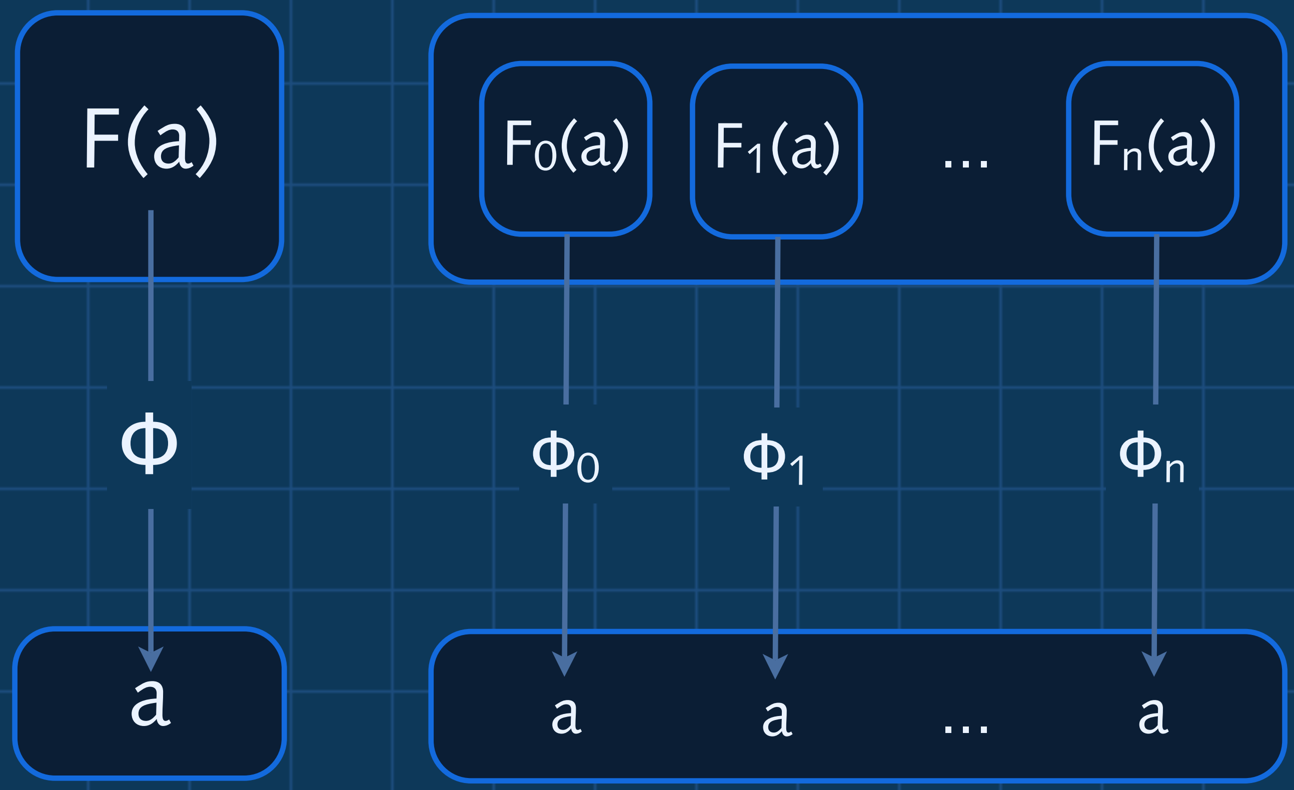
Processing Rosetta Specifications

- The Raskell frontend is a shared Parsec-based parser/printer
- AlgC automates boilerplate generation
- InterpreterLib defines semantic algebras and algebra combinators
  - Functors are implemented as semantic algebras
  - Facet algebra operations become algebra combinators or semantic algebras
  - Galois connections are used to assure transformation properties



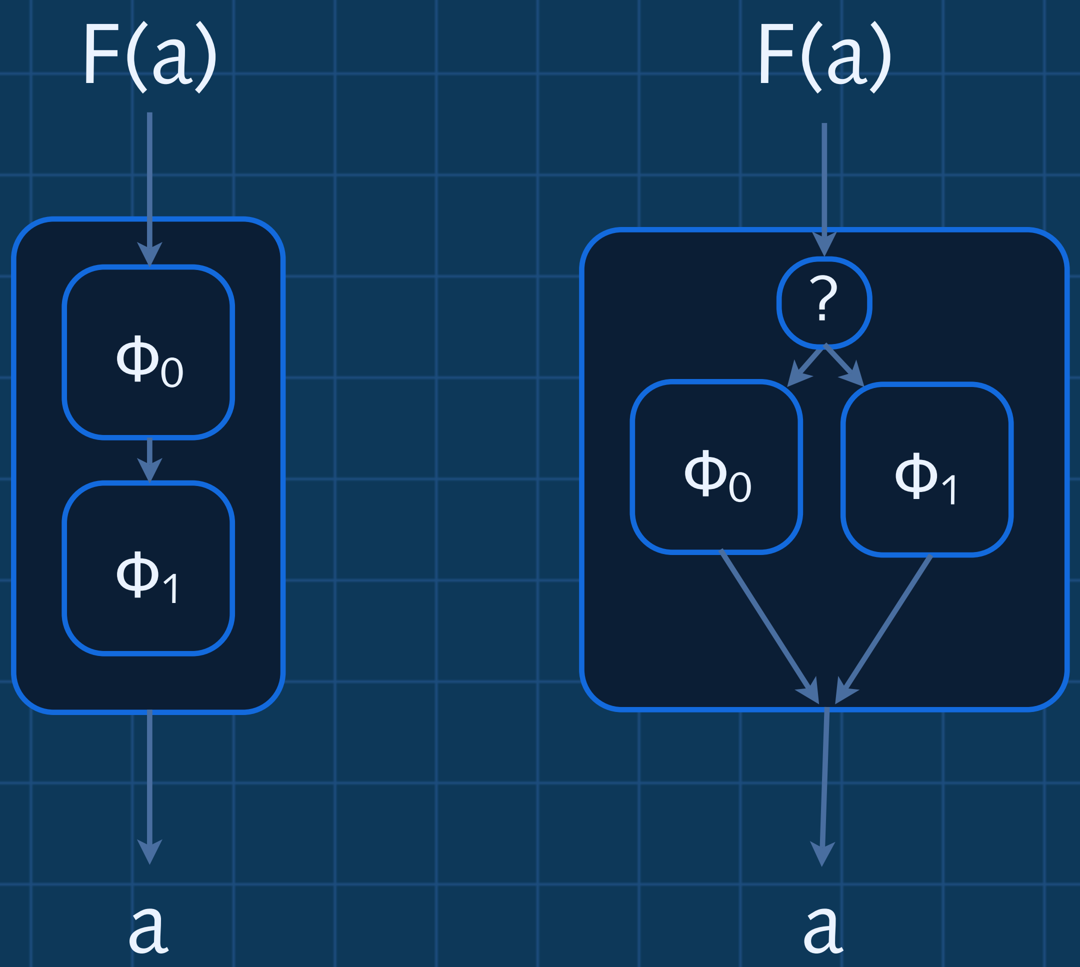
Semantic Algebras

- Principled mechanism for developing interpreters
  - Static analyzers
  - Language transformations
  - Traditional interprters
- Define a syntactic functor F
  - Define modular functors F_0-F_n
  - Compose to form F
- Define semantic algebra Φ
  - Define modular functors Φ_0-Φ_n
  - Compose to form Φ
- Use catamorphism to fold Φ into F(a)



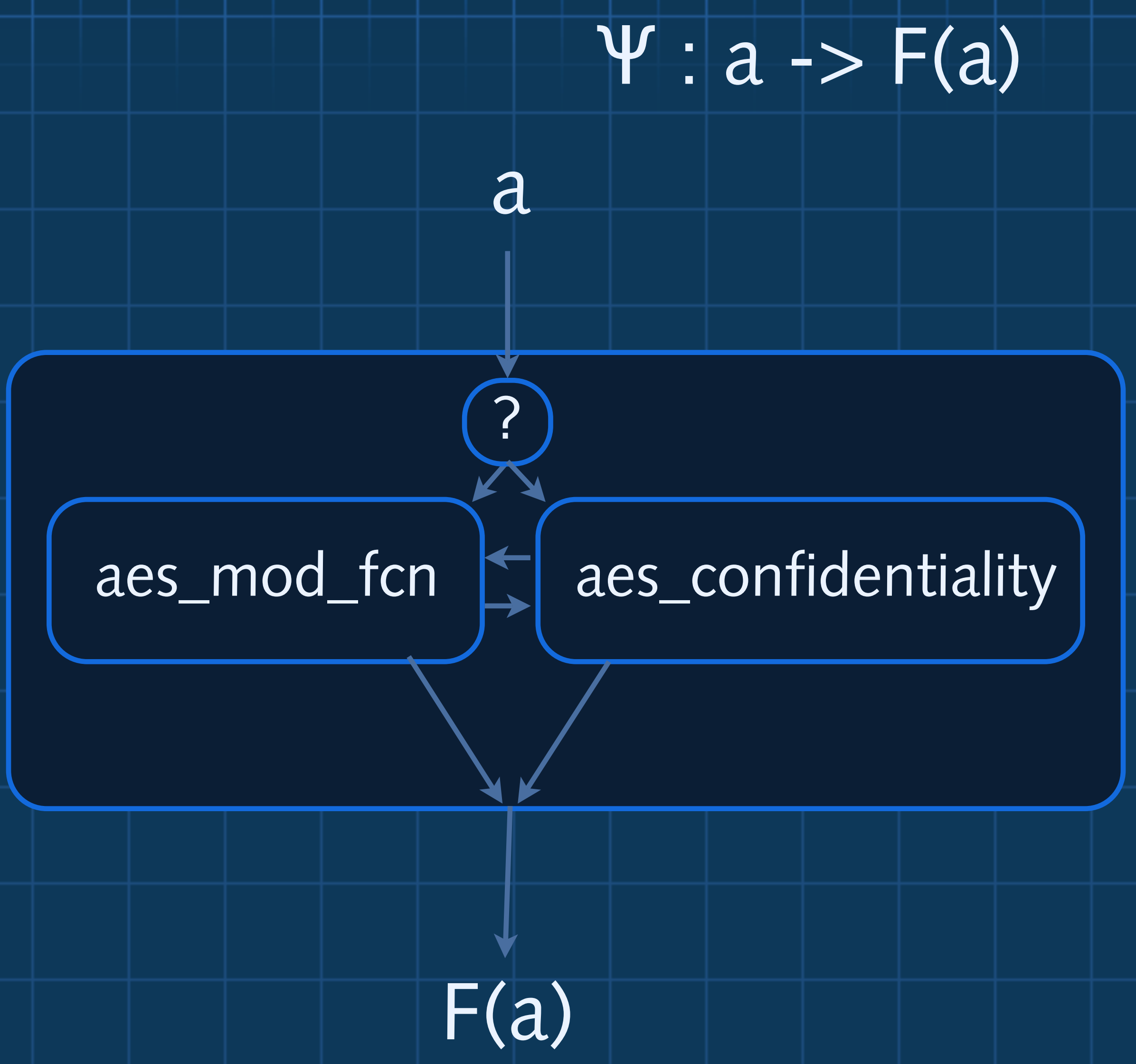
Algebra Combinators

- Composition principles for semantic algebras and comonadic simulators
- Sequence algebra sequences interpretations
  - Simple sequence
  - Paramorphism
- Switch algebra selects alternative interpretations
  - Modal interpretation
  - Parallel interpretation



Comonadic Simulators

- Rosetta facets and components are denoted as coalgebras
- If the carrier can be denoted as a comonad, a simulator results
- Coalgebraic simulators are composable like semantic algebras
  - Composed coalgebras are coalgebras
- Simulators can be analyzed in multiple ways
  - Formally using model checkers and theorem provers
  - Informally using traditional execution techniques



Current Status

- Rosetta Language Definition
  - Standard in preparation by IEEE DASC P1699 Rosetta Working Group (currently 70% complete)
  - Alexander, P., System-Level Design with Rosetta, Morgan Kaufmann Publishers, November 2006.
  - Alexander, P., System-Level Design Semantics, Morgan Kaufmann, Dec 2009 (in progress)
- Raskell
  - Parser, printer, recursive AST and non-recursive AST complete and usable
  - InterpreterLib and algc are complete and functional (GPCE'07, ASE'07 papers)
  - prototype composable, comonadic simulators are complete and usable (papers under review)
  - SAL and Isabelle interfaces being developed
  - Prototype Eclipse authoring and analysis module is available

Current Status

- Active Rosetta/Raskell Applications
  - Power-aware design
  - Software Defined Radio Synthesis
  - Secure system specification and analysis
  - Trust specification and analysis
- More information at http://www.rosetta-lang.org