Using Software Transformation Systems as Program Generator Backends

Ewen Denney        Johann Schumann
USRA/RIACS, NASA Ames

Bernd Fischer
ECS, U Southampton
Application domains

• Two main application domains
  – Guidance, Navigation & Control
  – Data Analysis

• Two common characteristics
  – Concise mathematical models
  – Algorithmic variability

• Top two algorithm families
  – Kalman Filters → AutoFilter
  – Clustering → AutoBayes

• Highly mathematical domains
Spacecraft, aircraft, ships, and (increasingly) cars require methods for the accurate determination of position and attitude

- **Equipment:**
  - Compass, clock, GPS, INS
  - Radio navigation (DME, Radar)

- **Problems:**
  - Measurements are noisy
  - Each measurement contributes partial information
  - Sensor failures (degradation, transient, permanent)

- **Overall task:**
  Calculate the best possible state estimate using all available information
Synthesis for GN&C

- Standard technology: Kalman filters
- Commercial autocoders insufficient
  - algorithmic variability
  - not adaptable
- Specialized generator for GN&C: AutoFilter
- High-level domain-specific modeling language
  - differential equations
- Supports model-based development
Kalman Filters: Model

**Process model**

- **State variables**
  - Process transition matrix: \( F_k \) (\( n \times n \))
  - Process noise variables: \( u_k \) \( \sim N(0, Q_k) \)
  - \( E[u_k u_i^T] = \begin{cases} Q_k, & i = k \\ 0, & i \neq k \end{cases} \)

- **Measurement variables**
  - Measurement sensitivity matrix: \( H_k \) (\( m \times n \))
  - Measurement noise variables: \( v_k \) \( \sim N(0, R_k) \)
  - \( E[v_k v_i^T] = \begin{cases} R_k, & i = k \\ 0, & i \neq k \end{cases} \)

\[ x_{k+1} = F_k x_k + G_k u_k \]

\[ z_k = H_k x_k + v_k \]

**Measurement model**

- **Process noise variables**
- **Measurement noise variables**
Kalman Filters: Model

**Process model**

\[ x_{k+1} = F_k x_k + G_k u_k \]

- Attitude, speed, etc.
- Flight dynamics
- Turbulences, headwind, etc.

**Measurement model**

\[ z_k = H_k x_k + v_k \]

- Altimeter, radar, barometer, etc.
- Instrument wiring
- Radar uncertainty, etc.
Kalman Filters: Model

**Process model**

\[ x_{k+1} = F_k x_k + G_k u_k \]

**Measurement model**

\[ z_k = H_k x_k + v_k \]

- Location, yaw, yaw rate
- Rover dynamics
- Wheel slippage
- Gyro uncertainty, etc.

- IMU, wheel odometry
- Sensors
Kalman Filters: Algorithm

1. Initialization: initialize all vectors and matrices.
   \[ K = P^{-1}H^{T} (HP^{-1}H^{T} + R)^{-1} \]

2. Measurement update: read and process measurement \( z \).
   \[ x^+ = x^- + K(z - Hx^-) \]
   \[ P^+ = (I - KH)P^- \]

   \[ x^- = Fx^+ \]
   \[ P^- = FP^+F^T + Q \]

4. Go to 2.
model mog as 'Mixture of Gaussians'.

const nat n_points.
where 0 < n_points.
const nat n_classes := 3.
where n_classes << n_points.
...
double mu(0..n_classes-1).
double sigma(0..n_classes-1).
where 0 < sigma(_).
...
data double x(0..n_points-1).
x(I) ~ gauss(mu(c(I)), sigma(c(I))).
max pr(x|{rho,mu,sigma})
for {rho,mu,sigma}.

- Schema library
- Symbolic subsystem
  - rewrite engine
  - symbolic differentiation
  - (polynomial) equation solver
- Procedural intermediate language
- Multiple backends
  - C/C++ based: Octave, Matlab, CLARAty
- Multiple programs synthesized
Schemas

- Algorithmic knowledge encoded as *schemas*
  - Schema = Conditions + Code fragment
  - Recursively composed
  - Progressive instantiation of solution
  - Generates platform-independent intermediate code

\[
\text{schema( max } P(U|V) \text{ wrt } V, \text{ Code\_fragment} ) : \text{-} \\
\ldots \\
\rightarrow \text{ Code\_fragment} = \\
beg\text{begin} \\
\langle \text{guess values for } c[i] \rangle \\
\text{for } i:=1 \text{ to } N \text{ do for } j:=1 \text{ to } M \text{ do } q[i,j] := 0; \\
\text{for } k:=1 \text{ to } N \text{ do } q[k,c[k]] := 1; \\
\text{while-converging}(V) \text{ do} \\
\langle \text{max } P({q,U}|V) \text{ wrt } V \rangle \\
q[i,j] := \langle \ldots \rangle \\
\text{end} \\
\text{end} \\
\text{(* applicability constraints *)} \\
\text{(* Initialize *)} \\
\text{(* M-step *)} \\
\text{(* E-step: calculate } P(q|\{U,V\}) \text{ *)} \\
\text{(* end while-converging *)}
\]
Schemas as Transformations

- **Big-step transformations**
  - horizontal (model decompositions / transformations)
  - vertical (domain-specific algorithms)

- **Implemented as combination of techniques**
  - meta-program (check conditions)
  - graph rewriting (transform model)
  - templates (represent code fragments)
Schema Hierarchy

- Filter architecture
  - Linearized Kalman filter (with nominal trajectory)
    - Fundamental solution matrix
      - Approximate matrix exponential
        - Taylor series expansion
        - Laplace transform
        - Numerical integration
  - Extended Kalman filter (no nominal trajectory)
    - Linearize over time (continuous → discrete)
      - Linearize over state space
        - Linearity check
          - Jacobians
            - Differentiate
  - Basic Kalman filter
    - Compute estimate
      - Propagate estimate
        - Interpolate estimate
          - UD-factorization
Rewrites as Transformations

Small-step transformations encoded as conditional rewrites: $C \Rightarrow L = R$

• Differentiation
• Discretization
• Taylor expansion
• Matrix identities
• Linearization of set of equations
• Approximations
• Trigonometry

(+ simple algebraic identities)
STSS for Optimizations (I)

Observation: schema-based program construction offers *opportunities* for optimization

- based on use of independent building blocks
  - loop fusion
- based on instantiation of building blocks
  - loop unrolling
  - strength reduction
  - scalarization
- based on repeated use of specification information
  - constant propagation
STS for Optimizations (II)

Observation: schema-based program construction offers *support* for optimization

- exploits knowledge available at synthesis time
  
  \[
  \text{schema}(\max f(X,Y) \text{ wrt } X, \text{ Prog}) : - \\
  \text{Prog} = \langle \text{numeric optimization routine} \rangle \\
  \rightarrow \text{can hoist } Y \text{ out of loops without dataflow analysis}
  \]

- similar in spirit to anticipatory optimization

- requires integration of optimization and synthesis
Conclusions

• Highly mathematical domains
  – rich structure for transformations

• Combination of synthesis and optimization promising
  – maximum effect requires tight integration