Refactoring-as-Propositions: Proved Refactoring of Hybrid Systems via Proved Refinements

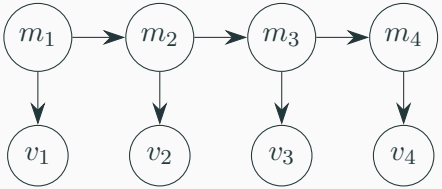
Enguerrand Prebet André Platzer

26th July 2026

Karlsruhe Institute of Technology

Cyber-Physical Systems and Refactoring-as-Propositions

Complex interactions of discrete and continuous behaviors

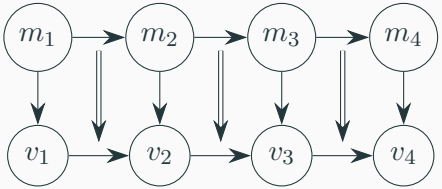


$\rightarrow$ iterative design

$\rightarrow$ repeated verification

Cyber-Physical Systems and Refactoring-as-Propositions

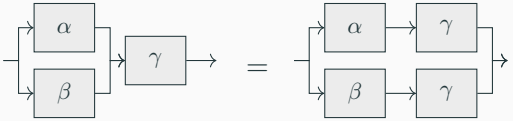
Complex interactions of discrete and continuous behaviors

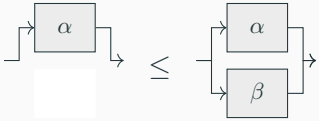


$\rightarrow$ iterative design

$\rightarrow$ repeated verification

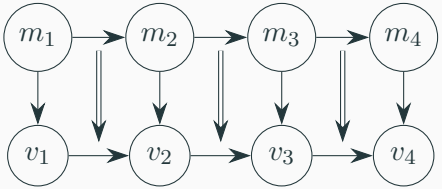
$\Rightarrow$ safe refactoring techniques





Cyber-Physical Systems and Refactoring-as-Propositions

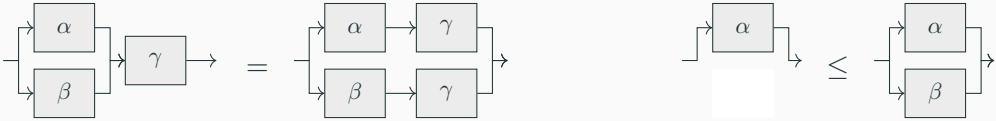
Complex interactions of discrete and continuous behaviors



$\rightarrow$ iterative design

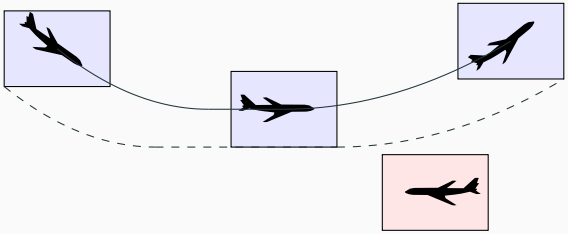
$\rightarrow$ repeated verification

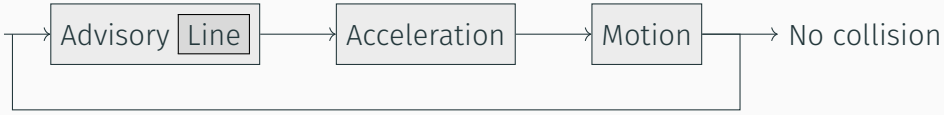
$\Rightarrow$ safe refactoring techniques



Refactoring-as-propositions are refinements: $\alpha \leq \beta$

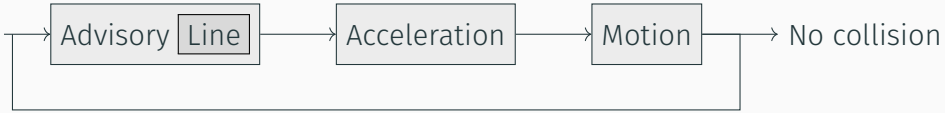
Example: Airborne Collision Avoidance System X[1]





[1] Jeannin et al. A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system. STTT 2017

Example: Airborne Collision Avoidance System X



Two versions of Line:

```
Bool case1(Real r, Real v, Real w, Real vlo, Real ru)  <-> -rp <= r & r < -rp - rv + minI(v,w)/alo;
Bool bound1(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> w = rv^2 + h < alo/2 * (r + rp)^2 - w * rv = v * (r+rp) - rv^2 + hp;
Bool case2(Real r, Real v, Real w, Real vlo, Real ru)  <-> -rp - rv = minI(v,w)/alo <= r & r <= rp - rv = minI(v,w)/alo;
Bool bound2(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> w = h < (-(minI(v,w)^2))/(2*alo) - hp;
Bool case3(Real r, Real v, Real w, Real vlo, Real ru)  <-> rp - rv = minI(v,w)/alo < r & r <= rp + rv = maxI(v,w,vlo)/alo;
Bool bound3(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> w = rv^2 + h < alo/2 * (r - rp)^2 + w = rv + v * (r - rp) - rv^2 + hp;
Bool case4(Real r, Real v, Real w, Real vlo, Real ru)  <-> rp + rv = maxI(v,w,vlo)/alo < r;
Bool bounds(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> ru < 0 | w < rv + h  :  w = vlo + (r - rp) - rv = maxI(v,w,vlo)^2/(2*alo) - rv + hp;
Bool case5(Real r, Real v, Real w, Real vlo, Real ru)  <-> -rp <= r & r < -rp + rv = maxI(v,w,vlo)/alo;
Bool bound5(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> w = rv^2 + h < alo/2 * (r + rp)^2 + w = rv + v * (r + rp) - rv^2 + hp;
Bool case6(Real r, Real v, Real w, Real vlo, Real ru)  <-> -rp < rv - rv = maxI(v,w,vlo)/alo <= r;
Bool bound6(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> ( (rv = 0 & r > rp)
                    | w = rv + h :  w = vlo + (r - rp) - rv = maxI(v,w,vlo)^2/(2*alo) - rv + hp );

Bool Lexp1(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> (
  (w + vlo >= 0 ->
     (case1(r,v,w,vlo,ru) -> bound1(r,h,v,w,vlo,ru))
   & (case2(r,v,w,vlo,ru) -> bound2(r,h,v,w,vlo,ru))
   & (case3(r,v,w,vlo,ru) -> bound3(r,h,v,w,vlo,ru))
   & (case4(r,v,w,vlo,ru) -> bound4(r,h,v,w,vlo,ru))
  )
  &
  (w + vlo < 0 ->
     (case5(r,v,w,vlo,ru) -> bound5(r,h,v,w,vlo,ru))
   & (case6(r,v,w,vlo,ru) -> bound6(r,h,v,w,vlo,ru))
  )
);
```

```
Bool Limp1(Real r, Real h, Real v, Real w, Real vlo, Real rv) <->
  \forall t. \forall ru !\forall ru (
    (0 <= t & t < maxI(v,w,vlo)/alo & ru = rv + t & hu = (w + alo)/2 * t^2 + v * t)
    | (t >= maxI(v,w,vlo)/alo & ru = rv + t & hu = vlo * t - w * maxI(v,w,vlo)^2/(2*alo))
    -> (abs(r - ru) > rp | w * h < w * hu - hp)
  );
```

Example: Airborne Collision Avoidance System X

Two versions of Line:

```
bool case1(Real r, Real v, Real w, Real vlo, Real ru)  <-> -rp <= r & r < -rp - rv * minI(v,w)/alo;
bool bound1(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> w = rv^2 + h < alo/2 * (r + rp)^2 + w * rv = v * (r+rp) - rv^2 + hp;
bool case2(Real r, Real v, Real w, Real vlo, Real ru)  <-> -rp - rv * minI(v,w)/alo <= r & r <= rp - rv * minI(v,w)/alo;
bool bound2(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> w = h < (-minI(v,w)^2)/(2*alo) - hp;
bool case3(Real r, Real v, Real w, Real vlo, Real ru)  <-> rp - rv * minI(v,w)/alo < r & r <= rp + rv * maxI(v,w,vlo)/alo;
bool bound3(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> w = rv^2 + h < alo/2 * (r - rp)^2 + w * rv = v * (r - rp) - rv^2 + hp;
bool case4(Real r, Real v, Real w, Real vlo, Real ru)  <-> rp + rv * maxI(v,w,vlo)/alo < r;
bool bounds(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> ru < 0 | w < rv + h  ∧ w + vlo < (r - rp) - rv * maxI(v,w,vlo)^2/(2*alo) - rv + hp;
bool case5(Real r, Real v, Real w, Real vlo, Real ru)  <-> -rp <= r & r < -rp + rv * maxI(v,w,vlo)/alo;
bool bound5(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> w = rv^2 + h < alo/2 * (r + rp)^2 + w * rv = v * (r + rp) - rv^2 + hp;
bool case6(Real r, Real v, Real w, Real vlo, Real ru)  <-> -rp < rv * maxI(v,w,vlo)/alo <= r;
bool bound6(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> ( (rv > 0 & r > rp)
                                  | w = rv + h ∧  w + vlo < (r - rp) - rv * maxI(v,w,vlo)^2/(2*alo) - rv + hp );

bool Lexp1(Real r, Real h, Real v, Real w, Real vlo, Real ru) <-> (
  (w + vlo >= 0 ->
     (case1(r,v,w,vlo,ru) -> bound1(r,h,v,w,vlo,ru))
   & (case2(r,v,w,vlo,ru) -> bound2(r,h,v,w,vlo,ru))
   & (case3(r,v,w,vlo,ru) -> bound3(r,h,v,w,vlo,ru))
   & (case4(r,v,w,vlo,ru) -> bound4(r,h,v,w,vlo,ru))
  )
  &
  (w + vlo < 0 ->
     (case5(r,v,w,vlo,ru) -> bound5(r,h,v,w,vlo,ru))
   & (case6(r,v,w,vlo,ru) -> bound6(r,h,v,w,vlo,ru))
  )
);
```

```
bool Limp1(Real r, Real h, Real w, Real vlo, Real rv) <->
  \forall t \forall ru \forall hu (
    (0 <= t & t < maxI(v,w,vlo)/alo & ru = rv + t & hu = (w + alo/2 * t^2 + v * t)
    | (t >= maxI(v,w,vlo)/alo & ru = rv + t & hu = vlo * t - w + maxI(v,w,vlo)^2/(2*alo))
    -> (abs(r - ru) > rp | w + h < w + hu - hp)
  );
```

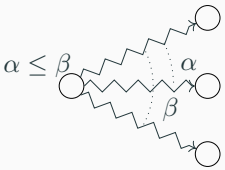
Safety transfer using refinements:

⇒ Manual steps reduced from 200 to 20

Refinements

$$\alpha \leq \beta$$

Meaning: β can reach all states that α can.

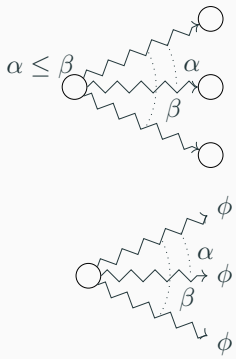


Refinements

$$\alpha \leq \beta$$

Meaning: β can reach all states that α can.

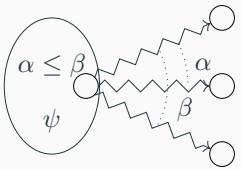
$$[\beta]\phi \to [\alpha]\phi$$



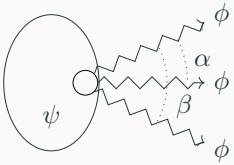
Refinements

$$\psi \to \alpha \leq \beta$$

Meaning: Starting from ψ, β can reach all states that α can.



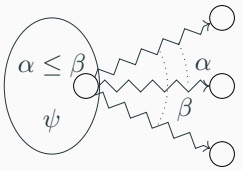
$$\psi \to ([\beta]\phi \to [\alpha]\phi)$$



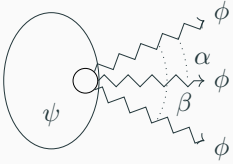
Refinements

$$\psi \to \alpha \leq \beta$$

Meaning: Starting from ψ, β can reach all states that α can.



$$\psi \to ([\beta]\phi \to [\alpha]\phi)$$



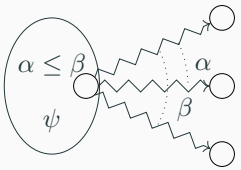
What about subsystems? Line

What about intermediate variables?

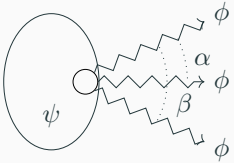
Refinements

$$\psi \to \alpha \leq \beta$$

Meaning: Starting from ψ, β can reach all states that α can.



$$\psi \to ([\beta]\phi \to [\alpha]\phi)$$



What about subsystems? Line

What about intermediate variables?

⇒ Automation solves both.

Differential Refinement Logic

Formulas and programs defined by mutual induction:

$$\phi, \psi ::= \theta \sim \eta \mid \neg\phi \mid \phi \wedge \psi \mid \forall x\, \phi \mid [\alpha]\phi \mid \alpha \leq \beta$$

$$\alpha, \beta ::= \underbrace{?\psi \mid x := \theta \mid x := *}_{\text{discrete}} \mid \underbrace{x' = \theta \,\&\, \psi}_{\text{continuous}} \mid \underbrace{\alpha \cup \beta \mid \alpha; \beta \mid \alpha^*}_{\text{composition}}$$

[1] Fulton et al. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. CADE 2015

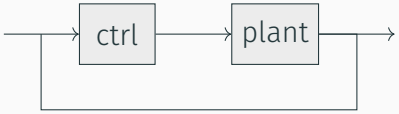
Differential Refinement Logic

Formulas and programs defined by mutual induction:

$$\phi, \psi ::= \theta \sim \eta \mid \neg\phi \mid \phi \wedge \psi \mid \forall x\, \phi \mid [\alpha]\phi \mid \alpha \leq \beta$$

$$\alpha, \beta ::= \underbrace{?\psi \mid x := \theta \mid x := *}_{\text{discrete}} \mid \underbrace{x' = \theta \,\&\, \psi}_{\text{continuous}} \mid \underbrace{\alpha \cup \beta \mid \alpha; \beta \mid \alpha^*}_{\text{composition}}$$

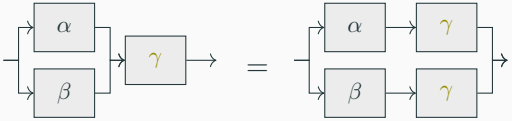
$(\text{ctrl}; \text{plant})^*$



Implemented in the KeYmaera X[1] theorem prover

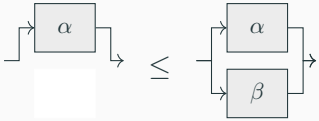
[1] Fulton et al. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. CADE 2015

Monotony for Global Refinements



$$(\alpha \cup \beta); \gamma \quad = \quad (\alpha; \gamma) \cup (\beta; \gamma)$$

$\Rightarrow$ original axiom [2]

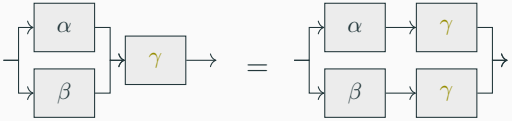


$$\alpha \quad \leq \quad \alpha \cup \beta$$

$\Rightarrow$ derived

[2] Prebet, Platzer. Uniform substitution for differential refinement logic. IJCAR 2024

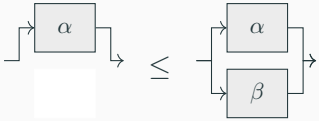
Monotony for Global Refinements



$$(\alpha \cup \beta); \gamma \quad = \quad (\alpha; \gamma) \cup (\beta; \gamma)$$

$\Rightarrow$ original axiom [2]

$$\frac{\alpha = \beta}{C(\alpha) = C(\beta)}$$



$$\alpha \quad \leq \quad \alpha \cup \beta$$

$\Rightarrow$ derived

$$\frac{\alpha \leq \beta}{C(\alpha) \leq C(\beta)}$$

Scalable: inference derivations automated

[2] Prebet, Platzer. Uniform substitution for differential refinement logic. IJCAR 2024

Monotony for Local Refinement

Context-aware refactoring: requires local information

$$\phi \to \alpha \leq \beta$$

Expected congruence principle:

$$\frac{\alpha \leq \beta}{C(\alpha) \leq C(\beta)}$$

Monotony for Local Refinement

Context-aware refactoring: requires local information

$$\phi \to \alpha \leq \beta$$

Expected congruence principle:

$$\frac{???(\phi) \quad \phi \to \alpha \leq \beta}{C(\alpha) \leq C(\beta)}$$

Monotony for Local Refinement

Context-aware refactoring: requires local information

$$\phi \to \alpha \leq \beta$$

Expected congruence principle:

$$\frac{\widetilde{C}_\alpha(\phi) \quad \phi \to \alpha \leq \beta}{C(\alpha) \leq C(\beta)}$$

$\Rightarrow \widetilde{C}_\alpha$ defined inductively over C

$$C(\sqcup) = (\gamma; \sqcup)^* \rightsquigarrow \widetilde{C}_\alpha(\sqcup) = [(\gamma; \alpha)^*][\gamma]\sqcup$$

Monotony for Local Refinement

Context-aware refactoring: requires local information

$$\phi \to \alpha \leq \beta$$

Expected congruence principle:

$$\frac{\widetilde{C}_\alpha(\phi) \quad \phi \to \alpha \leq \beta}{C(\alpha) \leq C(\beta)}$$

$\Rightarrow \widetilde{C}_\alpha$ defined inductively over C

$$C(\sqcup) = (\gamma; \sqcup)^* \rightsquigarrow \widetilde{C}_\alpha(\sqcup) = [(\gamma; \alpha)^*][\gamma]\sqcup$$

Result: Local refinement automatically ensures safety of whole systems ✓

Partial Refinement

Refactorings can introduce variables.

$\alpha \leq \beta$ captures all variables.

Partial Refinement

Refactorings can introduce variables.

$\alpha; x := * \leq \beta;$ $\underbrace{x := *}_{\substack{\text{nondet.} \\ \text{assignment}}}$ captures all variables but x.

$\Rightarrow$ Axioms are expressible.

Partial Refinement

Refactorings can introduce variables.

$\alpha; x := * \leq \beta;$ $\underbrace{x := *}_{\substack{\text{nondet.}\\ \text{assignment}}}$ captures all variables but x.

$\Rightarrow$ Axioms are expressible.

$$C(\alpha); x := * \quad = \quad C(\alpha; x := *); x := *$$

x fresh for C
$x \notin \mathsf{FV}(\alpha)$

Partial Refinement

Refactorings can introduce variables.

$\alpha; x := * \leq \beta;$ $\underbrace{x := *}_{\substack{\text{nondet.}\\ \text{assignment}}}$ captures all variables but x.

$\Rightarrow$ Axioms are expressible.

$$C(\alpha); x := * \quad = \quad C(\alpha; x := *); x := *$$

x fresh for C
$x \notin \mathsf{FV}(\alpha)$

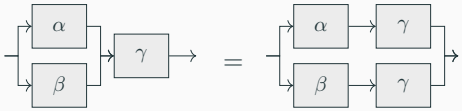
Result: equivalence derivation automated

Conclusion

Refinements are refactoring-as-propositions.

⇒ local (precondition)

⇒ partial (dismiss variables)



$$(\alpha \cup \beta); \gamma \quad = \quad (\alpha; \gamma) \cup (\beta; \gamma)$$

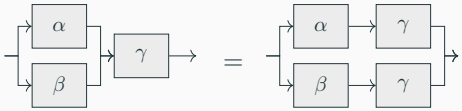
Conclusion

Refinements are refactoring-as-propositions.

⇒ local (precondition)

⇒ partial (dismiss variables)

⇒ with subsystem automatation



$$(\alpha \cup \beta); \gamma \quad = \quad (\alpha; \gamma) \cup (\beta; \gamma)$$

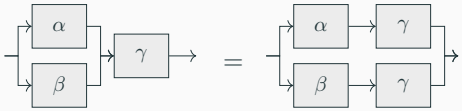
Conclusion

Refinements are refactoring-as-propositions.

⇒ local (precondition)

⇒ partial (dismiss variables)

⇒ with subsystem automatation



Thank you for listening!

References

Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer.
KeYmaera X: An axiomatic tactical theorem prover for hybrid systems.
In Amy Felty and Aart Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 527–538, Berlin, 2015. Springer.

Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Aurora Schmidt, Ryan Gardner, Stefan Mitsch, and André Platzer.
A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system.
STTT, 19(6):717–741, 2017.

Enguerrand Prebet and André Platzer.
Uniform substitution for differential refinement logic.
In Christoph Benzmüller, Marijn J. Heule, and Renate A. Schmidt, editors, *IJCAR*, volume 14740 of *LNCS*, pages 196–215. Springer, 2024.