1: Operators of Differential Dynamic Logic (dL)

dL	KeYmaera X	Operator	Meaning
$e = d$	`e=d`	equals	true if values of terms e and d are equal
$e \geq d$	`e>=d`	greater-or-equal	true if value of e greater-or-equal to value of d
$p(e_1,\dots,e_k)$	`p(e1,...,ek)`	predicate	true if p holds for the value of $(e1,\dots,ek)$
$\neg P$	`!P`	not / negation	true if P is false
$P \wedge Q$	`P & Q`	and / conjunction	true if both P and Q are true
$P \vee Q$	`P | Q`	or / disjunction	true if P is true or if Q is true
$P \rightarrow Q$	`P -> Q`	implies / implication	true if P is false or Q is true
$P \leftrightarrow Q$	`P <-> Q`	equivalent / bi-implication	true if P and Q are both true or both false
$\forall x P$	`\forall x P`	for all / universal quantifier	true if P is true for all real values of variable x
$\exists x P$	`\exists x P`	exists / existential quantifier	true if P is true for some real value of variable x
$[a]P$	`[a]P`	box / $[\cdot]$ modality	true if P is true after all runs of HP a
$\langle a\rangle P$	`<a>P`	diamond / $\langle\cdot\rangle$ modality	true if P is true after at least one run of HP a

Unary operators (including $\forall x, \exists x, [\alpha], \langle\alpha\rangle$) bind stronger than binary operators. And ; binds stronger than $\cup$.

2: Statements and effects of Hybrid Programs (HPs)

HP	KeYmaera X	Operation	Effect
$x := e$	`x:=e;`	discrete assignment	assigns value of term e to variable x
$x := *$	`x:=*;`	nondeterministic assign	assigns any real value to variable x
$x' = f(x) \,\&\, Q$	`{x'=f(x) & Q}`	continuous evolution	evolve along differential equation $x' = f(x)$ within evolution domain Q for any duration
$?Q$	`?Q;`	test	check first-order formula Q at current state
$a; b$	`a; b`	sequential composition	HP b starts after HP a finishes
$a \cup b$	`a ++ b`	nondeterministic choice	choice between alternatives HP a or HP b
a^*	`{a}*`	nondeterministic repetition	repeats HP a n-times for any $n \in \mathbb{N}$

```
Definitions        /* function symbols cannot change their value during any HP   */
  Real A = 5;      /* real-valued maximum acceleration constant is defined as 5 */
  Real B;          /* real-valued maximum braking constant is arbitrary          */
  Bool safe(Real v) <-> v>=0;        /* predicate of v for safety condition    */
  HP   accel      ::= {?v<=5; a:=A;}; /* subprogram for acceleration a:=A       */
End.

ProgramVariables   /* program variables may change their value over time */
  Real x, v;       /* real-valued position and velocity of a simple car  */
  Real a;          /* current acceleration chosen by the car controller  */
End.

Problem                              /* differential dynamic logic conjecture  */
    safe(v) & A>0 & B>0              /* initial condition where system starts  */
  ->                                 /* implies                                */
  [                                  /* all runs of hybrid program in [...]    */
    {                                /* braces {} for grouping HPs             */
      {accel; ++ a:=0; ++ a:=-B;}    /* nondeterministic choice acceleration a */
      {x'=v, v'=a & v>=0}            /* differential equation system in domain */
    }* @invariant(v>=0)              /* loop repeats, @invariant contract      */
  ] safe(v)                          /* safety/postcondition after HP          */
End.
```

3: Axioms

`assignb` $[:=]$ $[x:=e]p(x) \leftrightarrow p(e)$

`randomb` $[:*]$ $[x:=*]p(x) \leftrightarrow \forall x\, p(x)$

`testb` $[?]$ $[?Q]P \leftrightarrow (Q \rightarrow P)$

`solve` $[']$ $[x' = f(x) \,\&\, q(x)]p(x) \leftrightarrow \forall t{\geq}0 \big((\forall 0{\leq}s{\leq}t\, q(x(s))) \rightarrow [x:=x(t)]p(x)\big)$ (if $x'(t) = f(x(t))$)

`choiceb` $[\cup]$ $[a \cup b]P \leftrightarrow [a]P \wedge [b]P$

`composeb` $[;]$ $[a;b]P \leftrightarrow [a][b]P$

`iterateb` $[^*]$ $[a^*]P \leftrightarrow P \wedge [a][a^*]P$

`diamond` $\langle\cdot\rangle$ $\neg[a]\neg P \leftrightarrow \langle a\rangle P$

`K` K $[a](P \rightarrow Q) \rightarrow ([a]P \rightarrow [a]Q)$

`I` I $[a^*]P \leftrightarrow P \wedge [a^*](P \rightarrow [a]P)$

`V` V $p \rightarrow [a]p$ $(\mathrm{FV}(p) \cap \mathrm{BV}(a) = \emptyset)$

4: Differential equation sequent calculus proof rules

`dW` dW $\dfrac{\Gamma_{\text{const}}, Q \vdash p(x), \Delta_{\text{const}}}{\Gamma \vdash [x' = f(x) \,\&\, Q]p(x), \Delta}$

`dI` dI $\dfrac{\Gamma, Q \vdash P, \Delta \quad Q \vdash [x' := f(x)](P)'}{\Gamma \vdash [x' = f(x) \,\&\, Q]P, \Delta}$

`dC` dC $\dfrac{\Gamma \vdash [x' = f(x) \,\&\, Q]C, \Delta \quad \Gamma \vdash [x' = f(x) \,\&\, Q \wedge C]P, \Delta}{\Gamma \vdash [x' = f(x) \,\&\, Q]P, \Delta}$

`dG` dG $\dfrac{\Gamma \vdash \exists y\, [x' = f(x), y' = a(x)y + b(x) \,\&\, Q]P, \Delta}{\Gamma \vdash [x' = f(x) \,\&\, Q]P, \Delta}$ (postcondition P can be replaced by G if $G \vdash P$)

5: Propositional sequent calculus proof rules

`notR` $\neg$R $\dfrac{\Gamma, P \vdash \Delta}{\Gamma \vdash \neg P, \Delta}$

`notL` $\neg$L $\dfrac{\Gamma \vdash \Delta, P}{\neg P, \Gamma \vdash \Delta}$

`implyR` $\rightarrow$R $\dfrac{\Gamma, P \vdash \Delta, Q}{\Gamma \vdash P \rightarrow Q, \Delta}$

`implyL` $\rightarrow$L $\dfrac{\Gamma \vdash \Delta, P \quad Q, \Gamma \vdash \Delta}{P \rightarrow Q, \Gamma \vdash \Delta}$

`andR` $\wedge$R $\dfrac{\Gamma \vdash P, \Delta \quad \Gamma \vdash Q, \Delta}{\Gamma \vdash P \wedge Q, \Delta}$

`andL` $\wedge$L $\dfrac{\Gamma, P, Q \vdash \Delta}{P \wedge Q, \Gamma \vdash \Delta}$

`equivR` $\leftrightarrow$R $\dfrac{\Gamma, P \vdash \Delta, Q \quad \Gamma, Q \vdash \Delta, P}{\Gamma \vdash P \leftrightarrow Q, \Delta}$

`equivL` $\leftrightarrow$L $\dfrac{P \wedge Q, \Gamma \vdash \Delta \quad \neg P \wedge \neg Q, \Gamma \vdash \Delta}{P \leftrightarrow Q, \Gamma \vdash \Delta}$

`orR` $\vee$R $\dfrac{\Gamma \vdash \Delta, P, Q}{\Gamma \vdash P \vee Q, \Delta}$

`orL` $\vee$L $\dfrac{P, \Gamma \vdash \Delta \quad Q, \Gamma \vdash \Delta}{P \vee Q, \Gamma \vdash \Delta}$

`id` id $\dfrac{}{P, \Gamma \vdash P, \Delta}$

`cut` cut $\dfrac{\Gamma, C \vdash \Delta \quad \Gamma \vdash \Delta, C}{\Gamma \vdash \Delta}$

`closeTrue` $\top$R $\dfrac{}{\Gamma \vdash \mathit{true}, \Delta}$

`closeFalse` $\bot$L $\dfrac{}{\mathit{false}, \Gamma \vdash \Delta}$

`hideR` WR $\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash P, \Delta}$

`hideL` WL $\dfrac{\Gamma \vdash \Delta}{P, \Gamma \vdash \Delta}$

PR $\dfrac{\Gamma \vdash Q, P, \Delta}{\Gamma \vdash P, Q, \Delta}$

PL $\dfrac{Q, P, \Gamma \vdash \Delta}{P, Q, \Gamma \vdash \Delta}$

6: Quantifier sequent calculus proof rules

`allR` $\forall$R $\dfrac{\Gamma \vdash p(y), \Delta}{\Gamma \vdash \forall x\, p(x), \Delta}$ $(y \notin \Gamma, \Delta, \forall x\, p(x))$

`allL` $\forall$L $\dfrac{\Gamma, p(e) \vdash \Delta}{\Gamma, \forall x\, p(x) \vdash \Delta}$ (arbitrary term e)

`existsR` $\exists$R $\dfrac{\Gamma \vdash p(e), \Delta}{\Gamma \vdash \exists x\, p(x), \Delta}$ (arbitrary term e)

`existsL` $\exists$L $\dfrac{\Gamma, p(y) \vdash \Delta}{\Gamma, \exists x\, p(x) \vdash \Delta}$ $(y \notin \Gamma, \Delta, \exists x\, p(x))$

7: dL Sequent calculus proof rules

`loop` loop $\dfrac{\Gamma\vdash J,\Delta \quad J\vdash P \quad J\vdash [a]J}{\Gamma\vdash [a^*]P,\Delta}$

`generalize` MR $\dfrac{\Gamma\vdash [a]Q,\Delta \quad Q\vdash P}{\Gamma\vdash [a]P,\Delta}$

`generalize` ML $\dfrac{\Gamma,[a]Q\vdash \Delta \quad P\vdash Q}{\Gamma,[a]P\vdash \Delta}$

`abstract` GVR $\dfrac{\Gamma_{\text{const}}\vdash P,\Delta_{\text{const}}}{\Gamma\vdash [a]P,\Delta}$

`discreteGhost` iG $\dfrac{\Gamma\vdash [y:=e]p,\Delta}{\Gamma\vdash p,\Delta}$ (y new)

`CEat` CER $\dfrac{\Gamma\vdash C(Q),\Delta \quad \vdash Q\leftrightarrow P}{\Gamma\vdash C(P),\Delta}$

`CEat` CEL $\dfrac{\Gamma,C(Q)\vdash \Delta \quad \vdash Q\leftrightarrow P}{\Gamma,C(P)\vdash \Delta}$

`CEat` CQR $\dfrac{\Gamma\vdash p(k),\Delta \quad \vdash k=e}{\Gamma\vdash p(e),\Delta}$

`CEat` CQL $\dfrac{\Gamma,p(k)\vdash \Delta \quad \vdash k=e}{\Gamma,p(e)\vdash \Delta}$

CTR $\dfrac{\vdash e=k}{\Gamma\vdash c(e)=c(k),\Delta}$

CTL $\dfrac{\vdash e=k}{\Gamma,c(e)=c(k)\vdash \Delta}$

`US` US $\dfrac{\Gamma\vdash\Delta}{\sigma(\Gamma)\vdash\sigma(\Delta)}$ UR $\dfrac{\Gamma^y_x\vdash\Delta^y_x}{\Gamma\vdash\Delta}$ BRR $\dfrac{\Gamma\vdash [y:=e]\varphi^y_x,\Delta}{\Gamma\vdash [x:=e]\varphi,\Delta}$ BRL $\dfrac{\Gamma,[y:=e]\varphi^y_x\vdash\Delta}{\Gamma,[x:=e]\varphi\vdash\Delta}$ $(y,y',x'\not\in FV(\varphi))$

8: Differential equation axioms

`DW` DW $[x'=f(x)\,\&\,Q]P\leftrightarrow[x'=f(x)\,\&\,Q](Q\rightarrow P)$

`DI` DI $\big([x'=f(x)\,\&\,Q]P\leftrightarrow[?Q]P\big)\leftarrow(Q\rightarrow[x'=f(x)\,\&\,Q)](P)')$

`DC` DC $\big([x'=f(x)\,\&\,Q]P\leftrightarrow[x'=f(x)\,\&\,Q\wedge C]P\big)\leftarrow[x'=f(x)\,\&\,Q]C$

`DE` DE $[x'=f(x)\,\&\,Q]P\leftrightarrow[x'=f(x)\,\&\,Q][x':=f(x)]P$

`DG` DG $[x'=f(x)\,\&\,Q]P\leftrightarrow\exists y\,[x'=f(x),y'=a(x)y+b(x)\,\&\,Q]P$

9: First-order axioms

`allInst` $\forall$i $(\forall x\, p(x))\rightarrow p(e)$

`allDist` $\forall\!\rightarrow$ $\forall x\,(P\rightarrow Q)\rightarrow(\forall x\,P\rightarrow\forall x\,Q)$

`allV` $V_\forall$ $p\rightarrow\forall x\,p$ $(x\not\in FV(p))$

`existsDual` $\exists$ $\neg\forall x\neg P\leftrightarrow\exists x\,P$

10: Derived rules

`cohideR` WR $\dfrac{\vdash P}{\Gamma\vdash P,\Delta}$

`cohideL` WL $\dfrac{P\vdash}{P,\Gamma\vdash\Delta}$

`cohide2` WLR $\dfrac{P\vdash Q}{P,\Gamma\vdash Q,\Delta}$

`cutR` cutR $\dfrac{\Gamma\vdash Q,\Delta \quad \Gamma\vdash Q\rightarrow P,\Delta}{\Gamma\vdash P,\Delta}$

`cutL` cutL $\dfrac{Q,\Gamma\vdash\Delta \quad \Gamma\vdash\Delta,P\rightarrow Q}{P,\Gamma\vdash\Delta}$

`commuteEquivR` $\leftrightarrow c$R $\dfrac{\Gamma\vdash Q\leftrightarrow P,\Delta}{\Gamma\vdash P\leftrightarrow Q,\Delta}$

`commuteEquivL` $\leftrightarrow c$L $\dfrac{Q\leftrightarrow P,\Gamma\vdash\Delta}{P\leftrightarrow Q,\Gamma\vdash\Delta}$

`equivifyR` $\rightarrow 2\leftrightarrow$ $\dfrac{\Gamma\vdash P\leftrightarrow Q,\Delta}{\Gamma\vdash P\rightarrow Q,\Delta}$

11: Differential axioms

DS	DS $[x' = c() \,\&\, q(x)]p(x) \leftrightarrow \forall t{\geq}0\, \big((\forall 0{\leq}s{\leq}t\, q(x+c()s)) \rightarrow [x:=x+c()t]p(x)\big)$
Dconst	c' $(c())' = 0$
Dvar	x' $(x)' = x'$
Dplus	$+'$ $(e+k)' = (e)' + (k)'$
Dminus	$-'$ $(e-k)' = (e)' - (k)'$
Dtimes	$\cdot'$ $(e\cdot k)' = (e)'\cdot k + e\cdot (k)'$
Dquotient	$/'$ $(e/k)' = ((e)'\cdot k - e\cdot (k)')/k^2$
Dcompose	$\circ'$ $[y:=g(x)][y':=1]\big((f(g(x)))' = (f(y))'\cdot (g(x))'\big)$

12: Bellerophon tactic language operators for proof search

Bellerophon	Operation	Effect
`s ; t`	sequential composition	run `t` on the output of `s`, failing if either fail
`s | t`	alternative choice	run `t` if applying `s` failed, failing if both fail
`t*`	saturating repetition	repeat tactic `t` until nothing changes any more
`t*n`	fixed repetition	repeat tactic `t` exactly `n` times, failing if any of those repetitions fail
`<(t1,..,tn)`	branching	run tactic `ti` on branch `i`, failing if any fail or if branches $\neq$`n`
`l: t`	on branch	run tactic `t` on branch with label `l`
`doall(t)`	all branches	run tactic `t` on all branches `i`, failing if that fails on any branch
`t(j)`	at position	apply tactic `t` at position `j` of the sequent
`t(j,"e")`	at position	apply tactic `t` to expression `e`, which is at position `j` of the sequent
`1`	succedent position	position of first succedent formula. Similar `2`, `3`, ..., `'Rlast`
`-1`	antecedent position	position of first antecedent formula. Similar `-2`, `-3`, ..., `'Llast`
`-4.0.1`	subposition	second child of first child of fourth antecedent formula Similar `4.1`
`'R`	search succedent	first applicable succedent position (where formula `e` is, if specified)
`'L`	search antecedent	first applicable antecedent position (where formula `e` is, if specified)

```
/* Mix explicit and search tactic for above example instead of just tactic auto */
implyR(1) ; andL('L)* ; loop("v>=0", 1) ; <(       /* < splits separate branches */
  "Init": id,                          /* initial case: prove by identity v>=0|-v>=0 */
  "Post": QE,                          /* postcondition: prove by real arithmetic QE */
  "Step":                              /* induction step: decomposes HP explicitly   */
  composeb(1) ; solve(1.1) ; choiceb(1) ; andR(1) ; <(    /* controller branches */
    composeb(1) ; testb(1) ; auto,           /* decompose some steps then automate */
    choiceb(1) ; andR(1) ; doall(assignb(1) ; QE) /* doall same on all branches */
  )
)
```