

A Modular Programming Language for Engineering Design

by

Thomas Merritt Coffee

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2008

© Massachusetts Institute of Technology 2008. All rights reserved.

Author
Department of Aeronautics and Astronautics
February 1, 2008

Certified by
Annalisa L. Weigel
Assistant Professor
Thesis Supervisor

Accepted by
David L. Darmofal
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

A Modular Programming Language for Engineering Design

by

Thomas Merritt Coffee

Submitted to the Department of Aeronautics and Astronautics
on February 1, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

We introduce a new universal model of computation called MDPL that generalizes other functional models like the lambda calculus and combinatory logic. This model leads naturally to a new type of programming language that combines the key strengths of imperative and functional languages for development and analysis of programs. These strengths have particular relevance for rapid analysis of large-scale engineering design problems.

MDPL uses a novel approach to provide not only the flexibility to easily change relationships among elements in a program (as in imperative languages), but also the power to easily reuse and redeploy existing structures of such relationships in new places within a program (as in functional languages). The core formalism of MDPL is highly structured, but can be operated on by a family of formally defined algorithmic transformations that can automatically modify the structure of MDPL programs in useful ways to alter the relationships between essentially arbitrary program elements. These operations change the structure of a program to either change its functional interpretation, or to provide a different representation with the same interpretation, which may be used to make further changes. These algorithmic transformations play a critical role in rapid, incremental development of MDPL programs.

We describe a prototype implementation of an MDPL-based graphical programming environment targeted at engineering modeling tasks. This environment is used to conduct an experimental case study comparing the performance of the MDPL-based environment with a mainstream spreadsheet-based environment in the hands of engineers addressing a space systems design analysis task in a time-critical setting. The results of the case study illustrate some of the practical advantages of MDPL and confirm the intuition behind its design. We also discuss the theoretical capabilities of MDPL in relation to general-purpose computer programming, and explain how it captures or supersedes many of the important features of existing programming languages from multiple paradigms.

Finally, we discuss opportunities for future research and development, including: efficient implementation of the MDPL formalism; an effective graphical user interface for programming in MDPL-based languages; and potential extensions to MDPL for specialized application areas.

Thesis Supervisor: Annalisa L. Weigel

Title: Assistant Professor

Acknowledgments

My deepest thanks go to my beloved, Shannon Dong, for her endless counsel and courage during the past three years. This thesis would never have been completed without the strength of her conviction, and no words can express my appreciation for all that she has done. I can't wait to come home.

My parents, Peter Coffee and Carolyn Major, have been constantly there for me during this project and always. Since the day I was born they have done everything they can to make things possible for me. I only wish I could have spent more time with them during these past few years.

My thesis advisor, Annalisa Weigel, has been a pillar of patience and wisdom throughout the writing of this thesis. I have always left her office feeling better than when I came in.

The Lean Aerospace Initiative at MIT provided support for my research on integrated design teams, which became the motivation for this work. I will greatly miss the spirited and friendly people of the research group there.

I wish to thank Ross Jones, Becky Wheeler, and Luke Voss at the Jet Propulsion Laboratory for giving me the opportunity to work with Team X and pursue the practical side of my research at JPL. I am looking forward to spending much more time there in the future.

Several individuals from across the aerospace industry contributed their time to my early investigations of integrated design teams, which opened my eyes to many practical issues in engineering design. They will unfortunately remain anonymous, but their ideas have been a great help.

I also owe a great deal to Stephen Wolfram, Roger Germundsson, and all my colleagues at Wolfram Research, with whom I spent a wonderful summer in 2004. The things I learned from them have influenced every aspect of my research, and Mathematica has been an essential springboard for every part of this work.

My officemates Jason Bartolomei, Matt Richards, and Nirav Shah have always been eager to talk about my work and share their own ideas. Their friendly and collegial spirit will be sorely missed.

I must thank Ed Crawley for funding the project that became the precursor to this thesis. His leadership in supporting student projects is one of the greatest assets the department has.

My long-time mentor Oli de Weck provided some early feedback and encouragement on the latter project, and ultimately helped convince me to complete this work for my master's thesis.

In the early stages of this project, David Hartzband provided some useful comments and healthy skepticism on my research goals.

During my time at MIT, John Keesee has always provided sound words of advice, and has helped me forward from when the ideas in this project first formed until today.

My friend Ben Koo spent a great deal of time with me discussing ideas about computational modeling tools, and this work shares many of the same ultimate goals as the system he developed in his own thesis.

I owe Barbara Lechner and Marie Stuppard enormous thanks for their kind assistance in every difficulty I've encountered along my way through MIT. They have made this department a warm and welcoming place to work.

My friend Paul Wooster was the first person with whom I discussed the ideas that eventually became this project, and he encouraged me to take time away from our work on the Mars Gravity project to pursue them. His wisdom and leadership have been an inspiration to me throughout my time at MIT.

Finally, I wish to thank Eric Weisstein of Wolfram MathWorld and the many anonymous authors of Wikipedia for the tremendous references they provide on mathematics, computer science, and so much more. I believe they will eventually help the whole world to think more clearly.

“Technology is not neutral. Each technology has properties—affordances—that make it easier to do some activities, harder to do others: The easier ones get done, the harder ones neglected. Each has its constraints, preconditions, and side effects that impose requirements and changes on the things with which it interacts, be they other technology, people, or human society at large. Finally, each technology poses a mind-set, a way of thinking about it and the activities to which it is relevant, a mind-set that soon pervades those touched by it, often unwittingly, often unwillingly. The more successful and widespread the technology, the greater its impact upon the thought patterns of those who use it, and consequently, the greater its impact upon all of society. Technology is not neutral, it dominates.”

—Donald Norman, *Things that Make Us Smart*

Contents

1	Introduction	17
1.1	Frontiers of Engineering Design	17
1.2	Computer Programming	21
1.3	Related Work	23
1.4	Overview	26
2	Background	28
2.1	Polynomial Algebra	28
2.2	Graph Theory	29
2.3	Theory of Computation	35
3	The MDPL Language	38
3.1	Formal Definition	38
3.2	Visual Representation	40
3.3	Model Evaluation	43
3.4	Universal Computation	56
4	Model Transformations	63
4.1	Semantic Transformations	65
4.2	Syntactic Transformations	71
4.3	Partial Inversion Transformation	82
5	Theoretical & Experimental Results	86
5.1	Prototype Development Environment	86
5.2	Case Study: Space Telescope Architecture Analysis	93
5.3	General Features of the MDPL Language	125
6	Future Research & Development	129
6.1	User Interface Development	129

6.2	Efficient Implementation	131
6.3	Algebraic Transformations	132
6.4	Core Language Extensions	133
7	Conclusion	135
	Bibliography	137

List of Figures

1-1	The conceptual design phase commits a large fraction of the total resources of an engineering project based on a relatively small initial investment in design effort [25]	18
1-2	Integrated design teams like JPL's Team X conduct rapid conceptual design studies for spacecraft missions using models incorporating thousands of variable parameters [12]	19
3-1	An MDPL model representing the extended GCD function	42
3-2	The translation function \mathcal{L} from the sk combinator calculus to MDPL	57
3-3	Normalization of the MDPL translation of $((k X) Y)$	58
3-4	Normalizations of the MDPL translations of $((s X) Y) Z)$ (top) and $((X Z) (Y Z))$ (bottom).	59
3-5	Abbreviated normal order reduction of the MDPL translation of $((s k) k)$	60
3-6	Reductions involving a fixpoint model (highlighted) applied to an arbitrary model S	61
5-1	MDPL model computing the three roots of the cubic polynomial $ax^3 + bx^2 + cx + d$ over the complex numbers from the coefficients b , c , a , and d (in order from top to bottom), where i denotes the complex unit $\sqrt{-1}$ and a superscript n denotes taking the n th power	91
5-2	Expression trees corresponding to the three roots of the cubic polynomial $ax^3 + bx^2 + cx + d$ over the complex numbers in terms of coefficients a , b , c , and d , where i denotes the complex combination $u + iv$ of its two subelements u and v	92
5-3	Binary matrix relating values and equations to variables in the space telescope model, where $\bullet = 0$ and $\blacksquare = 1$	101
5-4	Block triangular decomposition of the matrix in Figure 5-3, where lines separate the corresponding row- and column-subsets	102
5-5	Spreadsheet implementation of the initial telescope model in Part I, where the highlighted formula corresponds to the equation $H_\lambda = \frac{2\pi \cdot h \cdot c^2}{\lambda^5} \cdot \frac{1}{\exp(\frac{hc}{\lambda kT}) - 1}$, and values shown are for the initial Earth-based RF design	103

5-6	MDPL implementation of the initial telescope model in Part I, where subnodes shown have submodel applicands implementing equations in the model corresponding to their labels	104
5-7	MDPL submodel applicand of the rightmost subnode in Figure 5-6, corresponding to the equation $SNR = \frac{N_p \cdot QE}{\sqrt{N_r^2 + N_p \cdot QE}}$	105
5-8	Spreadsheet expanding the implementation of Figure 5-5 over multiple wavelengths and locations	106
5-9	MDPL model applying the model in Figure 5-6 to multiple wavelengths and locations	107
5-10	Block triangular decomposition of the binary matrix relating parameters to values and equations in Part II.1	109
5-11	Submodel created by the partial inversion transformation containing the subnodes within a single component of the block triangular decomposition in Figure 5-10 . . .	111
5-12	Submodel replacing that in Figure 5-11 after partial inversion	113
5-13	Partial inversion of the model in Figure 5-6 by a permutation swapping $IFOV$ and T_i for SNR and C_T	115
5-14	Block triangular decomposition of the binary matrix relating parameters to values and equations in Part II.2	116
5-15	Log-linear plot of $8 \left(1 - e^{\frac{ch}{kT\lambda}}\right) r^2 N_p \lambda^4 + \pi^2 c D^2 R^2 \Delta \lambda T_i \tau_0 \tau_\lambda$ versus λ with initial values from the model in Part II.1: the rightmost root $\lambda = 1.0 \times 10^{-5}$ is the desired one	117
5-16	Log-linear-linear plot of $8 \left(1 - e^{\frac{ch}{kT\lambda}}\right) r^2 N_p \lambda^4 + \pi^2 c D^2 R^2 \Delta \lambda T_i \tau_0 \tau_\lambda$ versus λ and m_i with initial values from the model in Part II.1: the black arrow shows the movement of the desired root of λ as m_i moves from 39 174 kg to 60 000 kg	118
5-17	Model in Part III applying the original telescope model in two places to model a two-telescope system; note that despite the layout of the diagram, both subnodes on the bottom left are applicands of the same subnode above	119
5-18	Notes made by one participant using the spreadsheet environment in Part II.1, including diagrams similar to those automatically generated by the graphical representation of models in the MDPL environment	123
5-19	Log-log plot of Earth-based RF telescope total cost (C_T) versus integration time (T_i) by the only participant using the spreadsheet environment to produce plots in Part II.1	124

List of Tables

5.1	Model transformations applied by user interface actions in the prototype environment	88
5.2	Definitions and interpretations of MDPL primitives for list operations available in the prototype environment	90
5.4	Parameters with given values in Part I	96
5.6	Parameters with computed values in Part I	97
5.8	Results from the model in Part I	97
5.9	Additional parameter values in Part III	99
5.11	Results from the model in Part III	99

Nomenclature

Notation	Meaning
\equiv	equivalence (noncontextual)
$=$	equality (contextual)
\leftarrow	assignment (temporary)
\emptyset	undefined
\cdot	unspecified
$\mathbb{V}, \mathbb{W}, \dots$	named sets
$\mathcal{V}, \mathcal{W}, \dots$	named tuples
$\mathbf{V}, \mathbf{W}, \dots$	named sets of sets
$\mathcal{V}, \mathcal{W}, \dots$	named sets of tuples
$\mathfrak{V}, \mathfrak{W}, \dots$	named tuples of sets
$\mathbb{V}, \mathbb{W}, \dots$	named matrices
$\{v, w, \dots\}$	set with elements v, w, \dots ^a
$[v, w, \dots]$	tuple (list) with ordered elements v, w, \dots ^b
$\begin{bmatrix} v_{11} & v_{12} & \cdots \\ v_{21} & v_{22} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	matrix $[[v_{11}, v_{12}, \dots], [v_{21}, v_{22}, \dots], \dots]$
$(\mathcal{V}, \mathcal{W}, \dots)$	tuple $[\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_{ \mathcal{V} }, \mathcal{W}_1, \mathcal{W}_2, \dots, \mathcal{W}_{ \mathcal{W} }, \dots]$

^adefined as in Zermelo-Fraenkel set theory with the Axiom of Choice (ZFC) [78, 27, 61]: see Jech [34]

^bdefined based on the Wiener-Kuratowski ordered pair construction [77, 43]

Notation	Meaning
$\begin{pmatrix} \mathcal{V} \\ \mathcal{W} \\ \vdots \end{pmatrix}$	matrix $[\mathcal{V}, \mathcal{W}, \dots]$ where $ \mathcal{V} = \mathcal{W} = \dots$
$\begin{pmatrix} \mathbf{V} & \mathbf{W} & \dots \end{pmatrix}$	matrix $[(\mathbf{V}_1, \mathbf{W}_1, \dots), (\mathbf{V}_2, \mathbf{W}_2, \dots), \dots]$ where $ \mathbf{V} = \mathbf{W} = \dots$
$\begin{pmatrix} \mathbf{V} \\ \mathbf{W} \\ \vdots \end{pmatrix}$	matrix $[\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_{ \mathbf{V} }, \mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_{ \mathbf{W} }, \dots]$ where $ \mathbf{V}^T = \mathbf{W}^T = \dots$
$\begin{pmatrix} \mathbf{V}^{11} & \mathbf{V}^{12} & \dots \\ \mathbf{V}^{21} & \mathbf{V}^{22} & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}$	matrix $\begin{pmatrix} \begin{pmatrix} \mathbf{V}^{11} & \mathbf{V}^{12} & \dots \end{pmatrix} \\ \begin{pmatrix} \mathbf{V}^{21} & \mathbf{V}^{22} & \dots \end{pmatrix} \\ \vdots \end{pmatrix}$
$\langle \mathcal{V} \rangle$	set of all elements of \mathcal{V} : $\{\mathcal{V}_i\}_{i \in \langle n \rangle}$ where $n = \mathcal{V} $
$\langle \mathbf{V} \rangle$	sets of elements of elements of \mathbf{V} : $\{\langle \mathcal{V} \rangle\}_{\mathcal{V} \in \mathbf{V}}$
$\langle n \rangle$	tuple $[1, 2, \dots, n]$ where $n \in \mathbb{N}$
$\langle -n \rangle$	tuple $[-1, -2, \dots, -n]$ where $n \in \mathbb{N}$
$ n\rangle$	tuple $([0], \langle n \rangle)$
$\langle\langle n \rangle\rangle$	set $\{1, 2, \dots, n\}$ where $n \in \mathbb{N}$
$\mathbb{C}, \mathbb{R}, \mathbb{A}, \mathbb{Q}, \mathbb{Z}$	set of all complexes, reals, algebraics, rationals, integers, respectively ^c
\mathbb{Z}°	set of all nonzero integers: $\{z \in \mathbb{Z} \mid z \neq 0\}$ ^c
\mathbb{Z}^*	set of all nonnegative integers: $\{z \in \mathbb{Z} \mid z \geq 0\}$ ^c
\mathbb{Z}^+ or $\langle\langle \infty \rangle\rangle$ or \mathbb{N}	set of all positive integers (natural numbers): $\{z \in \mathbb{Z} \mid z > 0\}$ ^c
\mathbb{Z}^- or $\langle\langle -\infty \rangle\rangle$	set of all negative integers: $\{z \in \mathbb{Z} \mid z < 0\}$ ^c
\mathbb{B}	set of all booleans: $\{\top, \perp\}$
\mathbb{S}_p	set of all p -permutations: $\{f \in \mathbb{N}^{\mathbb{N}} \mid f : \langle\langle p \rangle\rangle \longleftrightarrow \langle\langle p \rangle\rangle\}$
\emptyset	empty set: $\{\}$
$\mathbf{0}$	matrix $\{[0]_{j \in \langle n \rangle}\}_{i \in \langle m \rangle}$ for some $m, n \in \mathbb{Z}^*$
$v \in \mathbb{V}$	v is an element of \mathbb{V} ^a

^cdefined as in standard complex analysis: see Rudin [58]

Notation	Meaning
$\mathbb{V} \in \mathbf{V}, \mathcal{V} \in \mathcal{V}$	\mathbb{V} is an element of \mathbf{V} , \mathcal{V} is an element of \mathcal{V} , respectively ^a
$v \in \mathcal{V}, \mathbb{V} \in \mathfrak{B}$	v is an element of \mathcal{V} , \mathbb{V} is an element of \mathfrak{B} : $v \in \langle \mathcal{V} \rangle$, $\mathbb{V} \in \langle \mathfrak{B} \rangle$, respectively ^b
$\mathcal{V} \in \mathfrak{B}$	each element of \mathcal{V} is an element of the corresponding element of \mathfrak{B} : $\mathcal{V} \in \prod \mathfrak{B}$
$\mathbb{U} \subseteq \mathbb{V}$	\mathbb{U} is a subset of \mathbb{V} : $\forall u \in \mathbb{U} u \in \mathbb{V}$
$\mathbb{U} \subseteq \mathcal{V}$	$\mathbb{U} \subseteq \langle \mathcal{V} \rangle$
$\mathcal{U} \subseteq \mathbb{V}, \mathcal{U} \subseteq \mathcal{V}$	$\langle \mathcal{U} \rangle \subseteq \mathbb{V}$, $\langle \mathcal{U} \rangle \subseteq \mathcal{V}$, respectively
$\mathcal{V} \subseteq \mathbb{V}$	\mathcal{V} orders \mathbb{V} : $\langle \mathcal{V} \rangle = \mathbb{V}$ and $ \mathcal{V} = \mathbb{V} $
$\mathcal{V} \subseteq \mathfrak{B}$	\mathcal{V} orders the elements of \mathfrak{B} : $\mathcal{V} = (\mathcal{V}^1, \mathcal{V}^2, \dots, \mathcal{V}^n)$ such that $\forall i \in \langle n \rangle \mathcal{V}^i \subseteq \mathfrak{B}_i$ where $n = \mathfrak{B} $
\mathcal{V}_i	i th element of \mathcal{V} , where $i \in \langle n \rangle$ and $n = \mathcal{V} $ ^b
$\mathcal{V}_{\mathcal{I}}$	subtuple $[\mathcal{V}_i]_{i \in \mathcal{I}}$ where $\mathcal{I} \subseteq \langle n \rangle$ and $n = \mathcal{V} $
$ \mathbb{V} , \mathcal{V} $	number of elements in \mathbb{V} , \mathcal{V} , respectively ^d
$ \mathcal{V} _v$	least index of v in \mathcal{V} : $\min\{k \in \langle n \rangle \mid \mathcal{V}_k = v\}$ where $n = \mathcal{V} $
$ \mathbb{V} $	number of rows in \mathbb{V}
$ \mathbb{V}^T $	number of columns in \mathbb{V}
$\mathbb{V}_{m \times n}$	matrix \mathbb{V} has m rows and n columns: $m = \mathbb{V} $ and $n = \mathbb{V}^T $
\mathbb{V}_i	i th row of \mathbb{V} , where $i \in \langle m \rangle$ and $m = \mathbb{V} $
\mathbb{V}_j^T	j th column of \mathbb{V} , where $j \in \langle n \rangle$ and $n = \mathbb{V}^T $
\mathbb{V}_{ij}	i, j entry of \mathbb{V} , where $i \in \langle m \rangle$, $j \in \langle n \rangle$, and $\mathbb{V}_{m \times n}$: $(\mathbb{V}_i)_j$
$\mathbb{V}_{\mathcal{I}\mathcal{J}}$	submatrix $[(\mathbb{V}_{ij})_{j \in \mathcal{J}}]_{i \in \mathcal{I}}$ where $\mathcal{I} \subseteq \langle m \rangle$, $\mathcal{J} \subseteq \langle n \rangle$, and $\mathbb{V}_{m \times n}$
$\mathbb{V}_{\mathbb{I}\mathbb{J}}$	submatrix $\mathbb{V}_{\mathcal{I}\mathcal{J}}$ for some $\mathcal{I} \subseteq \mathbb{I}$ and $\mathcal{J} \subseteq \mathbb{J}$ (unique up to row-column permutation)
\mathbb{V}^T	transpose of \mathbb{V} : matrix $(\mathbb{V}^T)_{n \times m}$ such that $\forall i \in \langle m \rangle \forall j \in \langle n \rangle \mathbb{V}_{ij} = \mathbb{V}_{ji}^T$ where $\mathbb{V}_{m \times n}$
$\mathcal{P}(\mathbb{V})$	power set of \mathbb{V} (set of all subsets of \mathbb{V}) ^a
$\mathcal{P}_k(\mathbb{V})$	set of all subsets of \mathbb{V} containing exactly k elements: $\{\mathbb{U} \in \mathcal{P}(\mathbb{V}) \mid k = \mathbb{U} \}$
$\mathbb{V} \setminus \mathbb{W}$	complement of \mathbb{W} in \mathbb{V} (set of all elements in \mathbb{V} not in \mathbb{W}): $\{v \in \mathbb{V} \mid v \notin \mathbb{W}\}$

^ddefined as equivalence classes of equipollence on bijective functions, following Tarski [65]: see Suppes [64]

Notation	Meaning
$\mathbb{V} \cup \mathbb{W} \cup \dots$	union of $\mathbb{V}, \mathbb{W}, \dots$ (set of all elements in any of $\mathbb{V}, \mathbb{W}, \dots$): $\cup\{\mathbb{V}, \mathbb{W}, \dots\}$
$\bigcup_{v \in \mathbb{V}} f(v)$	union of $f(v)$ over all $v \in \mathbb{V}$: $\bigcup f(\mathbb{V})$ where $f: \mathbb{V} \rightarrow \mathbb{W}$
$\bigcup \mathbf{V}$	union of sets in \mathbf{V} ^a
$\mathbb{V} \sqcup \mathbb{W} \sqcup \dots$	union $\mathbb{V} \cup \mathbb{W} \cup \dots$ sets $\mathbb{V}, \mathbb{W}, \dots$ are pairwise disjoint: $\bigsqcup\{\mathbb{V}, \mathbb{W}, \dots\}$
$\bigsqcup_{v \in \mathbb{V}} f(v)$	union $\bigcup_{v \in \mathbb{V}} f(v)$ sets in $f(\mathbb{V})$ are pairwise disjoint: $\bigsqcup f(\mathbb{V})$
$\bigsqcup \mathbf{V}$	union of sets in \mathbf{V} : $\bigsqcup_{v \in \mathbf{V}} \mathbb{V}$ sets in \mathbf{V} are pairwise disjoint: $\forall U \in \mathcal{P}_2(\mathbf{V}) \cap U = \emptyset$
$\mathbb{V} \oplus \mathbb{W} \oplus \dots$	exclusive union of $\mathbb{V}, \mathbb{W}, \dots$ (set of all elements in exactly one of $\mathbb{V}, \mathbb{W}, \dots$): $\oplus\{\mathbb{V}, \mathbb{W}, \dots\}$
$\oplus \mathbf{V}$	exclusive union of sets in \mathbf{V} : $\bigsqcup_{v \in \mathbf{V}} (\mathbb{V} \setminus \cup \mathbf{V} \setminus \{v\})$
$\mathbb{V} \cap \mathbb{W} \cap \dots$	intersection of $\mathbb{V}, \mathbb{W}, \dots$ (set of all elements in all of $\mathbb{V}, \mathbb{W}, \dots$): $\cap\{\mathbb{V}, \mathbb{W}, \dots\}$
$\bigcap_{v \in \mathbb{V}} f(v)$	intersection of $f(v)$ over all $v \in \mathbb{V}$: $\bigcap f(\mathbb{V})$
$\bigcap \mathbf{V}$	intersection of sets in \mathbf{V} : $\{v \in \bigcup \mathbf{V} \mid \forall v \in \mathbf{V} v \in \mathbb{V}\}$
$\mathbb{V} \times \mathbb{W} \times \dots$	direct product of $\mathbb{V}, \mathbb{W}, \dots$ (set of all tuples $[v, w, \dots]$ where $v \in \mathbb{V} \wedge w \in \mathbb{W} \wedge \dots$): $\prod[\mathbb{V}, \mathbb{W}, \dots]$
$\prod_{v \in \mathcal{V}} f(v)$	direct product of $f(v)$ over all $v \in \mathcal{V}$: $\prod f(\mathcal{V})$
$\prod \mathfrak{A}$	ordered direct product of sets in \mathfrak{A} ^a
$\mathbb{V} \otimes \mathbb{W} \otimes \dots$	exclusive direct product of $\mathbb{V}, \mathbb{W}, \dots$ (set of tuples of distinct elements in $\mathbb{V} \times \mathbb{W} \times \dots$): $\otimes[\mathbb{V}, \mathbb{W}, \dots]$
$\otimes \mathfrak{A}$	ordered exclusive direct product of sets in \mathfrak{A} : $\{\mathcal{V} \in \mathfrak{A} \mid \mathcal{V} \subseteq \langle \mathcal{V} \rangle\}$
\mathbb{V}^{\otimes}	set of 2-tuples of distinct elements of \mathbb{V} : $\mathbb{V} \otimes \mathbb{V}$
\mathbb{V}^n	set of n -tuples of elements of \mathbb{V} : $\prod_{i \in \langle n \rangle} \mathbb{V}$ where $n \in \mathbb{N}$
$\mathbb{V}^{m \times n}$	set of $m \times n$ matrices of elements of \mathbb{V} : $(\mathbb{V}^n)^m$
$f: \mathbb{V} \rightarrow \mathbb{W}$	(total) function f maps \mathbb{V} into \mathbb{W} : $\forall v \in \mathbb{V} \exists! w \in \mathbb{W} f(v) = w$
$f: \mathbb{V} \longleftrightarrow \mathbb{W}$	bijjective function f maps \mathbb{V} into \mathbb{W} : $f: \mathbb{V} \rightarrow \mathbb{W}$ and $\forall w \in \mathbb{W} \exists! v \in \mathbb{V} f(v) = w$

Notation	Meaning
$f(v)$	image $f(v) \in \mathbb{W}$ of $v \in \mathbb{V}$ under $f : \mathbb{V} \rightarrow \mathbb{W}$ where $f \subseteq \{[v, f(v)]\}_{v \in \mathbb{V}}$ ^e
$f(v, w, \dots)$	image $f([v, w, \dots]) \in \mathbb{W}$ of $[v, w, \dots] \in \mathbb{V}$ under $f : \prod \mathbb{V} \rightarrow \mathbb{W}$
$f(\mathbb{U})$ or $\{f(u)\}_{u \in \mathbb{U}}$	image $f(\mathbb{U}) \subseteq \mathbb{W}$ of $\mathbb{U} \subseteq \mathbb{V}$ under $f : \mathbb{V} \rightarrow \mathbb{W}$: $\{w \in \mathbb{W} \mid \exists u \in \mathbb{U} f(u) = w\}$
$f(\mathcal{V})$ or $[f(v)]_{v \in \mathcal{V}}$	image $\mathcal{W} \subseteq \mathbb{W}$ of $\mathcal{V} \subseteq \mathbb{V}$ under $f : \mathbb{V} \rightarrow \mathbb{W}$ such that $\forall_{i \in \langle n \rangle} \mathcal{W}_i = f(\mathcal{V}_i)$ where $ \mathcal{W} = \mathcal{V} = n$
$f \circ g$	composition of f and g : function $f \circ g : \mathbb{U} \rightarrow \mathbb{W}$ where $g : \mathbb{U} \rightarrow \mathbb{V}$, $f : \mathbb{V} \rightarrow \mathbb{W}$, and $\forall_{u \in \mathbb{U}} f \circ g(u) = f(g(u))$
$\bigcirc_{v \in \mathbb{V}} f(v)$	composition of $f(v)$ over all $v \in \mathbb{V}$: $\bigcirc f(\mathbb{V})$
$\bigcirc_{v \in \mathcal{V}} f(v)$	composition of $f(v)$ over all $v \in \mathbb{V}$: $\bigcirc f(\mathcal{V})$ for some $\mathcal{V} \subseteq \mathbb{V}$
$\bigcirc \mathcal{V}$	ordered composition of functions in \mathcal{V} : $\mathcal{V}_1 \circ \mathcal{V}_2 \circ \dots \circ \mathcal{V}_{ \mathcal{V} }$
f^n	n th-order composition of f : $\bigcirc_{i \in \langle n \rangle} f$ where $f : \mathbb{V} \rightarrow \mathbb{V}$ and $n \in \mathbb{N}$
f^{-1}	inverse function of f : function $f^{-1} : f(\mathbb{V}) \rightarrow \mathbb{V}$ such that $\forall_{v \in \mathbb{V}} f^{-1}(f(v)) = v$, where $f : \mathbb{V} \rightarrow f(\mathbb{V})$
$\mathbb{V}^{\mathbb{W}}$	set of all functions on \mathbb{V} that map \mathbb{V} into \mathbb{W} : $\{f \in \mathcal{P}(\mathbb{V} \times \mathbb{W}) \mid f : \mathbb{V} \rightarrow \mathbb{W}\}$
$\mathfrak{F}^{\mathfrak{W}}$	set of all functions on $\prod \mathfrak{V}$ that map $\prod \mathfrak{V}$ into $\prod \mathfrak{W}$: $\{f \in \mathcal{P}(\prod \mathfrak{V} \times \prod \mathfrak{W}) \mid f : \prod \mathfrak{V} \rightarrow \prod \mathfrak{W}\}$
\top, \perp	logical true, false, respectively ^f
$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$	logical not, and, or, implies, if and only if, respectively ^f
$\forall_{v \in \mathbb{V}} q(v)$	$q(v) = \top$ for all $v \in \mathbb{V}$ where $q : \mathbb{V} \rightarrow \mathbb{B}$ ^g
$\exists_{v \in \mathbb{V}} q(v)$	$q(v) = \top$ for some $v \in \mathbb{V}$ where $q : \mathbb{V} \rightarrow \mathbb{B}$: $\neg \forall_{v \in \mathbb{V}} \neg q(v)$ ^g
$\exists!_{v \in \mathbb{V}} q(v)$	$q(v) = \top$ for exactly one $v \in \mathbb{V}$ where $q : \mathbb{V} \rightarrow \mathbb{B}$: $\exists_{v \in \mathbb{V}} q(v) \wedge \forall_{v \in \mathbb{V}} \forall_{w \in \mathbb{V}} q(v) \wedge q(w) \Rightarrow v = w$ ^g
$\nexists_{v \in \mathbb{V}} q(v)$	$q(v) = \top$ for no $v \in \mathbb{V}$ where $q : \mathbb{V} \rightarrow \mathbb{B}$: $\forall_{v \in \mathbb{V}} \neg q(v)$
$\{v \in \mathbb{V} \mid q(v)\}$	set $\mathbb{U} \subseteq \mathbb{V}$ such that $\forall_{v \in \mathbb{V}} v \in \mathbb{U} \Leftrightarrow q(v)$ where $q : \mathbb{V} \rightarrow \mathbb{B}$ ^a
$\{f(v)\}_{v \in \mathbb{V} \mid q(v)}$	$\{f(u)\}_{u \in \{v \in \mathbb{V} \mid q(v)\}}$
$\{f(v) \mid q(v)\}$	$\{f(u)\}_{u \in \{v \mid q(v)\}}$

^edefined as sets of ordered pairs with unique images, following Dirichlet [21]: see Knopp [41]

^fdefined as in propositional logic: see Mendelson [50]

^gdefined as in first-order predicate logic: see Kleene [40]

Notation	Meaning
$\cdot_{v \in \mathbb{V} q(v)} f(v)$	$\cdot_{u \in \{v \in \mathbb{V} q(v)\}} f(u)$
$\lceil x \rceil$	ceiling of x (least integer not less than x): $\min\{z \in \mathbb{Z} \mid z \geq x\}$ where $x \in \mathbb{R}^c$
$\lfloor x \rfloor$	floor of x (greatest integer not greater than x): $\max\{z \in \mathbb{Z} \mid z \leq x\}$ where $x \in \mathbb{R}^c$
$\mathcal{O}(f(v, w, \dots))$	set of functions in the order of f : $\{g \in (\mathbb{R}^n)^{\mathbb{R}} \mid \limsup_{v \rightarrow \infty} \limsup_{w \rightarrow \infty} \dots \left \frac{g(v, w, \dots)}{f(v, w, \dots)} \right < \infty\}$ where $f : \mathbb{R}^n \rightarrow \mathbb{R}^c$
$\left\{ \begin{array}{ll} f_1(v) & q_1(v) \\ \vdots & \vdots \\ f_{n-1}(v) & q_{n-1}(v) \\ f_n(v) & q_n(v) \end{array} \right.$	<p>image $f(v)$ of $v \in \mathbb{V}$ under piecewise function $f : \mathbb{V} \rightarrow \mathbb{W}$ such that</p> <p>$\forall_{i \in \langle n \rangle} \forall_{u \in \mathbb{U}_i} f(u) = f_i(u)$ where $\forall_{i \in \langle n \rangle} f_i : \mathbb{U}_i \rightarrow \mathbb{W}$ and $\forall_{i \in \langle n \rangle} q_i : \mathbb{V}_i \rightarrow \mathbb{B}$ and</p> <p>$\forall_{i \in \langle n \rangle} \mathbb{U}_i = \{v \in \mathbb{V}_i \mid q_i(v) = \top\}$ and $\forall_{i \in \langle n \rangle} \mathbb{V}_{i+1} = \mathbb{V}_i \setminus \mathbb{U}_i$ and $\mathbb{V}_1 = \mathbb{V}$ and $\mathbb{V}_{n+1} = \emptyset$</p>
$\left\{ \begin{array}{ll} f_1(v) & q_1(v) \\ \vdots & \vdots \\ f_{n-1}(v) & q_{n-1}(v) \\ f_n(v) & \text{otherwise} \end{array} \right.$	$\left\{ \begin{array}{ll} f_1(v) & q_1(v) \\ \vdots & \vdots \\ f_{n-1}(v) & q_{n-1}(v) \\ f_n(v) & q_n(v) \end{array} \right.$

Chapter 1

Introduction

This work is motivated by challenges facing the conceptual design engineering of large space systems, activities with huge leverage in technological advancement and business development for many aerospace enterprises. To address these challenges, we begin by rethinking fundamental approaches to computer programming, creating a new programming language combining the flexibility and power of approaches that exist only separately today. Implementing this language involves applying a number of methods from technical computing together in novel ways to create a streamlined environment for conceptual design analysis.

1.1 Frontiers of Engineering Design

The conceptual design phase of engineering projects offers a tremendous opportunity for emerging information technologies to impact the successful completion of engineering projects. The conceptual design phase of a typical engineering project commits a substantial fraction of its lifecycle cost; however, the resources expended in conceptual design constitute a tiny fraction of this cost (Figure 1-1). As the spacecraft industry becomes more competitive, aerospace industry organizations are recognizing that more effective conceptual design processes are essential to maintaining their future business. Organizations able to explore more mission concepts more accurately in less time are better able to target business opportunities and win proposals.

With modern technology, competitive conceptual design efforts in the spacecraft industry require performing rapid analysis (on the order of days to weeks from concept formulation to proposal) with rather large design models (containing up to tens of thousands of variable parameters) [62, 60, 32, 4, 1]. These scale and time constraints pose steep challenges for automated tradespace exploration, and demand tight guidance from human engineers in the conceptual design process [55, 57, 75]. To take advantage of combined interdisciplinary expertise, aerospace companies have created integrated design centers like JPL's Team X that bring together engineers across the organization for intense

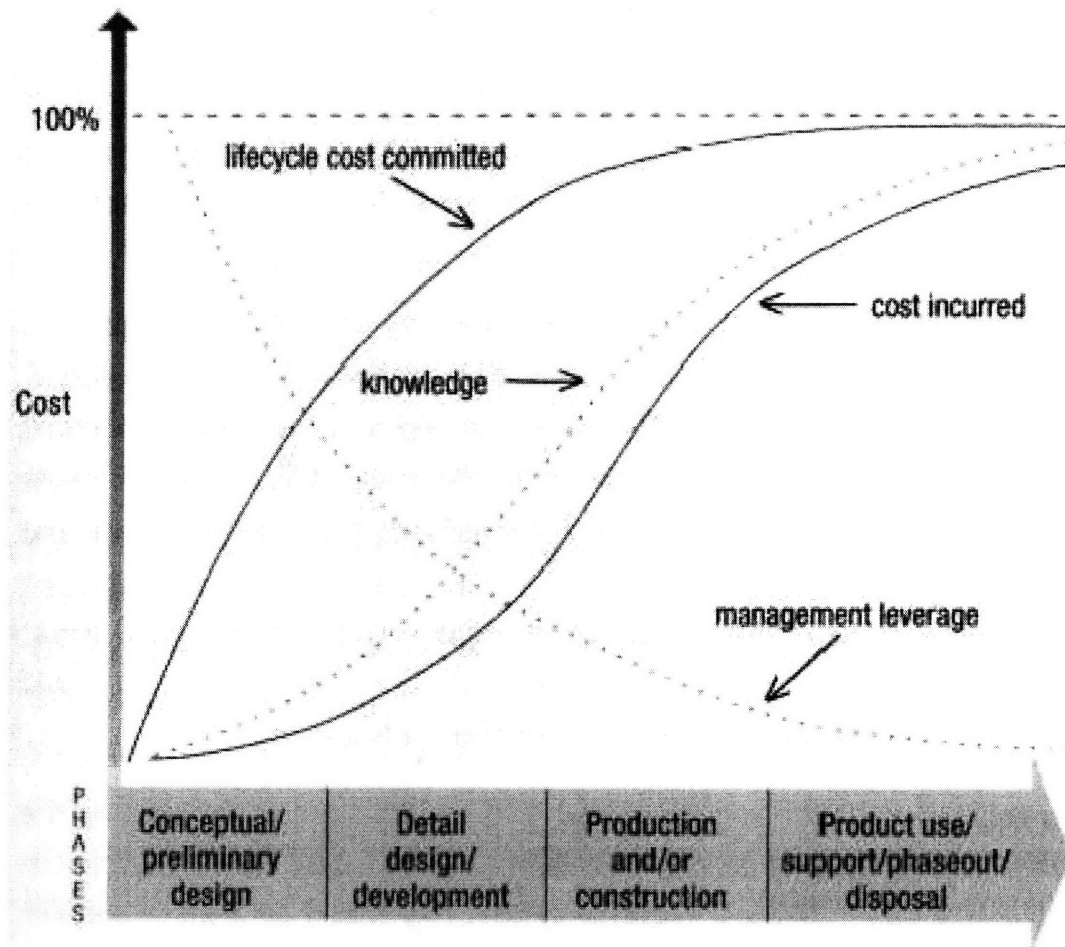


Figure 1-1: The conceptual design phase commits a large fraction of the total resources of an engineering project based on a relatively small initial investment in design effort [25]

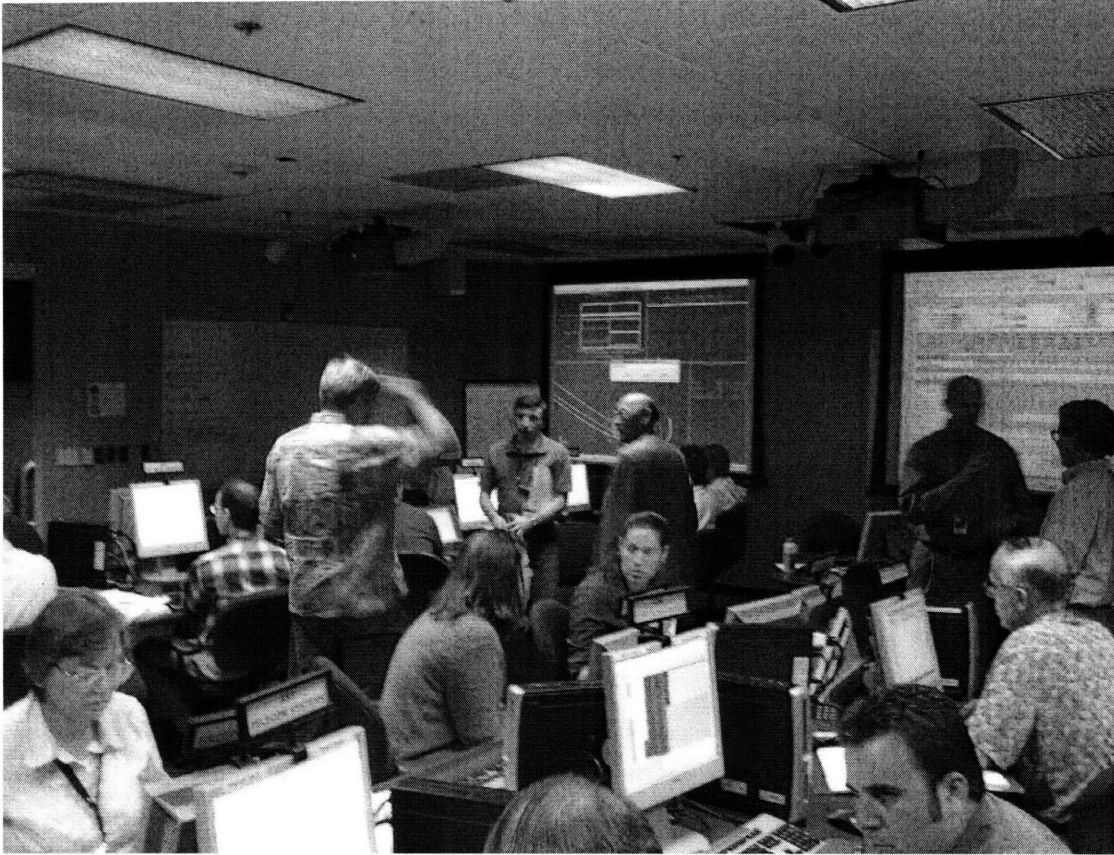


Figure 1-2: Integrated design teams like JPL's Team X conduct rapid conceptual design studies for spacecraft missions using models incorporating thousands of variable parameters [12]

design sessions working in real time with a homegrown family of modeling tools. Ideally, the design engineers are both users and developers of the modeling tools, creating continuous improvement in the organization's conceptual design capabilities.

The current state-of-the-art conceptual design centers in the spacecraft industry rely on large modeling tools built on networked versions of the Microsoft[®] Excel[®] spreadsheet software [52, 49, 12]. These modeling environments provide certain user interface advantages essential for dealing with large-scale models in rapid, collaborative settings: in particular, the values of many parameters at once are easily visible and changeable, providing each engineer an immediate picture of the current state of the system design. Some changes can be made on the fly to functions implementing parts of the model; however, these functions are not as easily visible and changeable, and moreover, the structure of their relationships is quite rigidly tied to the layout of the spreadsheet. Hence, while it is straightforward to explore changes to values of parameters within the model, it is much more difficult to explore structural changes to the model, as might correspond to alternative architecture options.

The limited architectural flexibility of modeling tools, combined with the pressure of rapid,

resource-constrained conceptual design efforts, leads to an inevitable strategy for industrial design centers. These centers invest substantial resources developing tools that span a certain space of system architectures closely aligned with the organization's core competencies (e.g., Earth-orbiting satellites). They then streamline these tools to optimize the conceptual design process for that space of architectures, which serves most needs of the organization well. Over time, they may pursue additional tool development projects to expand the center's capabilities into new architecture areas (e.g., optical instrument payloads), often creating new corresponding design teams. However, such tool development projects must be separately budgeted and cannot make use of the everyday operational resources of the design center. This strategy discourages design engineers from making changes to models on the fly, and reinforces the mindset that the tools are static entities, further eroding a key advantage of integrated design teams.

Moreover, inflexible modeling tools lead to problems at the boundaries of an organization's conceptual design capability, precisely where design centers have the greatest opportunity for a competitive edge. For example, the author observed a study at one industry design center analyzing a mission concept employing multiple probes that would jettison from a primary spacecraft after reaching a target area. The design center's models were built to deal with single-spacecraft missions, and while they had the necessary functionality to model all aspects of a dumb probe (trajectory, power, structure, communications, etc.), the structure of the model could not simultaneously accommodate multiple vehicles sharing additional design relationships between them. In the end, the design team performed a serial analysis in which the primary spacecraft was modeled first, then the models were reset and filled out with different parameters to effectively analyze the spacecraft-like probes. However, this serial approach impeded the ability to explore system-level design options affecting both the primary spacecraft and the probes.

To fully realize the potential of integrated design teams in the conceptual design process, these teams need modeling tools with greater inherent flexibility, that allow them to easily make changes not only to the values of parameters in a model, but to the structure of the model itself. This will enable teams to explore system architecture options as quickly as they explore system design options within a predefined architecture, and take better advantage of the intuition and expertise of the design engineers working to develop new mission concepts.

Exercising this kind of control effectively requires engineers to become programmers working with large programs under tight schedules, and demands new ways of interacting with programs beyond traditional programming languages. This thesis builds the foundations of a new type of programming language called MDPL, that gives engineers new forms of access and control over the functions relating the parameters within their models, as they now have over the parameters fed into their models. MDPL addresses fundamental challenges in the theory and practice of computing that have affected the development of programming languages since their inception.

1.2 Computer Programming

Computer programming was pioneered by two styles of programming languages, commonly called “imperative” and “functional” styles. Each resembles one of the two models of computation put forward in 1936 to answer Hilbert’s Entscheidungsproblem, the Turing machine of Alan Turing, and the lambda calculus of Alonzo Church [67, 11]. The imperative languages represent computation as the execution of a series of commands on a set of memory registers, resembling a Turing machine with a “random-access” tape. The functional languages represent computation as the evaluation of expressions consisting of nested functions, resembling a version of the lambda calculus. The canonical imperative languages are the assembly languages of microprocessors, while the canonical functional languages are pure LISP and its close relatives [48].

Over time, new languages have evolved that merge aspects of imperative and functional languages. Even languages that are considered representative of one style incorporate aspects of the other. For instance, the C language, generally considered an imperative language, allows one to write a statement containing nested mathematical expressions, and to define a function executing a set of statements based on the arguments of the function [39]. On the other hand, the common LISP language, generally considered a functional language, allows one to make assignments to named variables that persist across the boundaries of function evaluation [63]. More recent languages have combined these aspects even more freely: the “object-oriented” languages represent computation by evaluation of localized functions (functional style) of objects that execute commands on the objects’ local data (imperative style) [19, 18].

Imperative and functional styles each have advantages and disadvantages for the development and analysis of programs. Here “development” refers to the construction of programs by a programmer, often involving a great deal of iteration: the development process benefits from the ability to accomplish desirable changes to a program with minimal effort. In contrast, “analysis” refers to the understanding of existing programs, either by a programmer or a machine (e.g., a compiler): the analysis process benefits from the ability to easily determine useful properties of a program’s behavior (e.g., to improve the efficiency of its compilation).

In development, the imperative style gives the programmer more flexibility in that s/he may anywhere call upon and/or modify any value in memory, whereas the functional style limits the programmer’s access to values that have been passed to the function at hand, possibly requiring that s/he rewrite an unbounded number of neighboring functions in order to access to the one desired. On the other hand, the functional style gives the programmer more power in that s/he may define an arbitrarily large expression as a function and pass it to be evaluated anywhere, perhaps repeatedly, whereas the imperative style requires the programmer to delicately construct an arbitrarily large series of commands, perhaps temporarily reassigning an arbitrarily large number of values in memory, to take advantage of a sequence of operations defined elsewhere.

In analysis, the imperative style is more straightforward to translate into lower-level commands for machine execution, since its representation more closely resembles that used in typical machine hardware. Historically, this has facilitated development of compilers for imperative languages on a wide variety of machines, an important reason for their popularity. However, the purely functional style (due to its lack of flexibility) places many more constraints on the structure of programs, which can greatly aid both human and machine comprehension. In the past two decades, compilers have emerged for pure functional languages that take advantage of these constraints to automatically verify, optimize, and parallelize these programs for greater reliability and efficiency.

These advantages and disadvantages are familiar in programming practice. Imperative languages are well-suited to writing small, very efficient routines for high-performance computing. On the other hand, they can be intensely painful to debug, and unless carefully written, depend heavily on sophisticated compilers with advanced analysis capabilities for their efficiency. In contrast, functional languages are well-suited for prototyping large or mathematically involved systems quickly without much effort in debugging. But traditionally, they have made high efficiency difficult to achieve.

To the extent that languages merge aspects of the imperative and functional styles, they tend to lose the advantages of one or the other. In development, an imperative program containing large functional expressions or modules cannot anywhere access elements of those expressions or modules without rewriting those portions of the program, while a functional program containing commands that break functional boundaries cannot call upon that functionality elsewhere without going through the same difficulties as those faced by an imperative program. In analysis, imperative languages must be augmented with more advanced compilers to gain powerful functional features like passing functions, while impure functional languages incorporating imperative features quickly break the simplifying constraints on which they depend for their advantages. In spite of this, the most common languages combine aspects of both styles, often interleaving them with one another, perhaps to give the programmer freedom to approach any task with the most appropriate style within a “single” language.

The recent rise in popularity of pure functional languages is the combined result of ongoing research and ever-increasing computing resources. A steady decline in the cost of computing has long been responsible for the incorporation of more and more aspects of functional style into the most popular languages, as the financial trade-off between efficient development and efficient execution increasingly favors the former. At the same time, research into automated type-checking and compilation (among other areas) has produced a new breed of pure functional languages with all the advantages of the functional style but with strong software tools for automated verification and optimization. These advances are beginning to address the disadvantages of functional languages on the side of analysis.

However, even the most modern pure functional languages cannot match all the advantages of

imperative programming on the side of development: the structure imposed by the functional style still impedes the expression of algorithms. Languages like Haskell have developed some very sophisticated techniques to deal with some specific issues (for instance, monads for sequencing operations on state variables), but these do not approach the generic flexibility of the imperative style.

The MDPL language developed herein combines the advantages of the functional and imperative styles by representing programs in a purely functional way, but providing general algorithmic transformations that replicate the flexibility one finds in imperative programming (Chapter 4). These transformations allow the programmer immediate access to any value anywhere in the program, automatically modifying the functional structure of the program as necessary to achieve the desired effect. Moreover, the transformations permit the functional boundaries of elements of the program to be modified just as easily, so that the powerful features of the functional style can be immediately exploited anywhere in the program without prior effort. MDPL thus combines the creative flexibility of the imperative style with the intuitive and tractable structure of the functional style.

MDPL's use of a native graphical representation (§3.2) enables it to avoid the problems associated with named variables in symbolic languages, achieving some of the same advantages of so-called "tacit programming" in "function-level" programming languages like J [3]. These languages, like MDPL, resemble combinatory logic more closely than lambda calculus, since they eschew abstraction of named variables in favor of combining function structures directly to define new functions. Such languages have no need for the scoping rules analogous to α -conversion in lambda calculus, which allows them to avoid interpretation issues like the well-known FUNARG problem for traditional functional languages [51]. MDPL similarly avoids these issues while retaining the flexibility that named variables provide in program development.

In conceptual design, the structure provided by MDPL allows engineers to better understand the character of the architecture they are modeling and identify components of it that may be redeployed to change the architecture. The flexibility provided by MDPL allows them to quickly analyze those changes using elements of the model they have already developed. With an appropriate interface, engineers can perform these kinds of programming tasks as part of the normal process of design analysis, even under severe time constraints (Chapter 5). While we focus on engineering design tasks, these same benefits may apply to general computer programming.

1.3 Related Work

The methods developed herein bear some similarities to methods used elsewhere in the literature and in practice, some in active areas of research. We summarize here some related work in various domains.

Graphical Programming MDPL makes use of a graphical representation (§3.2) of programs based on hierarchical acyclic digraphs (§2.2.2), in which vertices represent the MDPL equivalent of functions, and edges represent values passed between functions. In fact, the rules governing the model of computation on which MDPL is based (Chapter 3) are most easily explained using this representation. Similar graphical representations have been used epistemologically to explain other models of computation such as the lambda calculus [38].

Graphical languages have been developed for decades in a variety of programming styles, and MDPL takes full advantage of the generic benefits of reading and writing programs in visual rather than symbolic form. These include rapid interpretation of the structure of a program, real-time verification of some aspects of syntactic correctness, and ease of selecting and manipulating logically distinct elements of a program [56]. However, the graphical representation of MDPL, combined with the features of the underlying language, provides capabilities well beyond those of existing graphical languages.

Graphical languages in the functional style have not seen widespread use. Multiple authors have developed graphical versions of LISP that provide some of the generic benefits above (along with some challenges in visual display and user interface) [23]. Because of the functional style of the underlying language, these representations bear some similarity to that of MDPL in that functions are represented as visual objects and connections between functions indicate the passing of values between them. However, these representations are not used to enhance or augment the structure of the LISP language. In particular, like all symbolic functional languages they restrict structural relationships to strict expression trees (Figure 5-2) rather than the more general nested acyclic digraphs of MDPL, and lack the transformations that MDPL provides to facilitate structural changes to programs.

In the imperative style, the best known graphical language is the G language used in the LabVIEW[®] software from National Instruments [30, 31]. Like common symbolic imperative languages, this language incorporates some functional elements like user-defined functions; however, its representation of programs emphasizes the serial execution of commands on data rather than the evaluation of structured expressions. Hence, while the representation uses visual objects and connections to indicate functions and passing of values, respectively, connections also indicate the control flow of execution within a program. Moreover, the facilities for manipulating functional elements have limitations similar to those of symbolic imperative languages: user-defined functions are not handled like other data values as in functional languages, but must be separately defined and called by name. Besides limiting expressive power, this makes the main facilities of graphical programming irrelevant to manipulating the functional structure of programs as in MDPL.

Graphical languages have also been developed for some special-purpose applications relevant to engineering design. In the area of system modeling, some integration tools like ModelCenter[®] from

Phoenix Integration use graphical languages in a more functional style to pass values between user-defined functions including external programs [46]. In engineering, these higher-level programs are used to perform trade analysis on parameter values within a statically defined system architecture, addressing some of the same applications as MDPL. However, such tools rely on external programs as “black boxes” to provide the computationally sophisticated elements of system models, and cannot take advantage of their underlying mathematical structure to perform trade analysis, as can MDPL (§4.3). Moreover, lacking the function manipulation facilities of a complete functional programming language, such tools do not provide the capability to easily explore variations in system architecture.

In the area of simulation, graphical languages like Simulink[®] from The MathWorks facilitate encoding simulation programs using nested digraphs, a visual representation even more general than the nested acyclic digraphs of MDPL [47]. However, the interpretation of such a representation is entirely different, since connections represent unbounded time-dependent data streams rather than finite data values. Such programs are executed by solving corresponding systems of discrete-continuous differential equations rather than evaluating functional expressions. They have a more specialized role within the domain of general computer programming addressed by MDPL.

Acyclic digraph representations similar to that of MDPL’s visual representation have also been used indirectly in the compilation of other programming languages. The graph data structures used in compiling some modern pure functional languages are closely analogous to the graphs generated in the evaluation of MDPL programs (§3.3) [74]. Similar data structures are often used in compiler analysis for optimizing imperative-language programs based on static single assignment form [16]. By using such structures in its native representation, MDPL can naturally take advantage of these approaches for machine execution (§6.2).

Computer Algebra The partial inversion transformation (§4.3) used to interchange the role of function inputs and outputs in certain classes of MDPL programs makes use of multiple techniques related to existing methods in computer algebra. These techniques are combined in novel ways to provide a more efficient, integrated, scalable approach to partial inversion than that available in existing tools.

The acyclic digraph representation of MDPL programs can encode many useful classes of mathematical functions more compactly than the more constrained expression trees of traditional functional languages, which leads to more efficient execution of results. For instance, the solutions of univariate polynomial systems have a very compact representation in MDPL compared with the corresponding functional expressions (Figures 5-1 & 5-2 show the case of cubic polynomials). Such compact representations are similar to the optimized encodings generated by methods like common subexpression elimination and tree height reduction in compiler and hardware design [44, 8, 45, 9]. These often have the added benefit of increased numerical stability [53].

The partial inversion transformation relies on a general form of sparse matrix block triangular decomposition to identify components of MDPL programs that must be solved simultaneously. This approach has seen widespread use in linear system solvers [59], but is not typically used in routines for solving nonlinear equations, which can easily lead to computational intractability in large systems. In addition, the MDPL environment implemented in Chapter 5 dynamically maintains the block triangular decomposition corresponding to the program under development by incrementally applying subroutines within the well-known block triangularization algorithm, leading to tremendous savings over multiple partial inversion transformations compared with a black-box approach. To the author’s knowledge, the latter approach has not been previously applied elsewhere.

In algebraically intractable cases, the partial inversion transformation relies on seminumerical techniques that use known solutions for initial values of equation variables to propagate solutions for other variable values based on error tolerances computed from numerical derivatives. These techniques involve methods similar to those used in computer algebra for initializing partially determined systems, as common, for instance, in numerical solvers for systems of differential-algebraic equations. This is a very active area of research: some relevant recent investigations are discussed in §6.3.

1.4 Overview

This work introduces a new model of computation closely related to other functional models like the lambda calculus and combinatory logic. This model leads naturally to a new type of programming language called MDPL, which combines the key strengths of imperative and functional languages for development and analysis of programs. These strengths have particular relevance for rapid analysis of large-scale engineering design problems.

Chapter 3 gives the formal specification of MDPL. We describe the correspondence between MDPL programs (called *models*) and computable functions by defining the evaluation of MDPL models in terms of normalizing transformation rules. Finally, we show that MDPL is universal (in the Church-Turing sense), and therefore capable of expressing all computable functions.

Chapter 4 defines a family of algorithmic transformations that can be used to automatically modify the structure of MDPL models in certain ways. These operations change the structure of a model to either change the functional interpretation of a model, or to provide a different representation with the same interpretation, which may be used to make further changes. Model transformations play a critical role in powerful, rapid development of programs.

Chapter 5 describes a prototype implementation of an MDPL modeling environment, with some extensions to the core language to aid engineering design work. We present an experimental case study comparing the performance of the MDPL environment with current design analysis tools in

the hands of engineers addressing a real-world space systems design task in a time-critical setting. Finally, we discuss the theoretical significance of MDPL's unique feature set, and speculate on its benefits for general-purpose computer programming.

Chapter 6 discusses some directions for future research and development relevant to MDPL. We sketch some approaches for the practical and efficient implementation of the MDPL language based on current and ongoing research in other programming languages and software tools. We also describe some practical extensions that will facilitate user interaction in an MDPL programming environment, some of which require further mathematical research and/or software development. We also refer the reader to ongoing research in mathematics and computer science that may further improve the techniques presented for the partial inversion transformation in (§4.3). Finally, we discuss the historical significance of MDPL among existing programming languages and modeling tools, and the potential for future extensions to the language and its capabilities.

Chapter 2

Background

This chapter introduces some theoretical background material underlying the remainder of the text. Our main purpose is to provide unambiguous terminology for ideas whose definitions or notation vary in the existing literature. The results presented here derive from the works of other authors, though the presentation may differ.

2.1 Polynomial Algebra

Some key results from polynomial algebra are important in the specialization of the partial inversion transformation to polynomial equations, discussed in §4.3.

2.1.1 Monomials and Polynomials

A *monomial* in the variables $\mathbb{X} = \{x_i\}_{i \in \langle n \rangle}$ for $n \in \mathbb{N}$ is a product of the form $\prod_{i=1}^n x_i^{\mathcal{K}_i}$ for $\mathcal{K} \in (\mathbb{Z}^*)^n$. The set of all monomials in the variables \mathbb{X} is denoted \mathbb{X}^\bullet . A *polynomial* of degree $m \in \mathbb{Z}^*$ in the variable x over a field \mathbb{K} is a sum of the form $\sum_{k \in \langle m \rangle} a_k x^k$ where the *coefficients* $a_k \in \mathbb{K}$ for each k . The set of polynomials of degree m in x over \mathbb{K} over all $m \in \mathbb{Z}^*$ is denoted by $\mathbb{K}[x]$.

A polynomial of total degree $m \in \mathbb{Z}^*$ in the variables \mathbb{X} is a sum of monomials in \mathbb{X} of the form $\sum_{\mathcal{K} \in (\mathbb{Z}^*)^n | \sum_{i \in \langle n \rangle} \mathcal{K}_i \leq m} a_{\mathcal{K}} \prod_{i=1}^n x_i^{\mathcal{K}_i}$ where the coefficients $a_{\mathcal{K}} \in \mathbb{K}$ for each \mathcal{K} . Each summand $a_{\mathcal{K}} \prod_{i=1}^n x_i^{\mathcal{K}_i}$ in the preceding summation is called a *term* of the resulting polynomial. The set of polynomials of total degree m in \mathbb{X} over \mathbb{K} over all $m \in \mathbb{Z}^*$ is denoted $\mathbb{K}[\mathbb{X}] = \mathbb{K}[x_1][x_2] \cdots [x_n]$. In what follows, we may omit the variables in which or field over which polynomials are specified, with the understanding that a consistent set of variables and field of coefficients is assumed.

A *polynomial ideal* in \mathbb{X} over \mathbb{K} is a set $\mathbb{H} \subseteq \mathbb{K}[\mathbb{X}]$ that is closed under multiplication by polynomials in $\mathbb{K}[\mathbb{X}]$, $\forall P \in \mathbb{K}[\mathbb{X}] \forall Q \in \mathbb{H} P \cdot Q \in \mathbb{H}$. For a set \mathbb{M} of polynomials in \mathbb{X} over \mathbb{K} , the ideal $\Psi(\mathbb{M}) \equiv \bigcup_{P \in \mathbb{K}[\mathbb{X}]} \{P \cdot Q\}_{Q \in \mathbb{M}}$ is called the ideal *generated* by \mathbb{M} , and \mathbb{M} is called a *basis* of $\Psi(\mathbb{M})$.

When the set \mathbb{M} is finite, $|\mathbb{M}| < \infty$, the ideal $\Psi(\mathbb{M})$ is called *finitely generated*.

2.1.2 Gröbner Bases

A *monomial order* on the variables \mathbb{X} is a well order $<_M$ on \mathbb{X}^\bullet for which $\forall U \in \mathbb{X}^\bullet \forall V \in \mathbb{X}^\bullet \forall W \in \mathbb{X}^\bullet U <_M V \Rightarrow U \cdot W <_M V \cdot W$. The *lexicographic order* on \mathbb{X} is the monomial order $<_L$ that orders monomials by the exponent of x_1 , then the exponent of x_2 , and so on, so that $U <_L V$ for monomials $U = \prod_{i=1}^n x_i^{\mathcal{I}_i}$ and $V = \prod_{i=1}^n x_i^{\mathcal{J}_i}$ when $\exists i \in \langle n \rangle \mathcal{I}_i < \mathcal{J}_i \wedge \forall j \in \langle i-1 \rangle \mathcal{I}_j = \mathcal{J}_j$.

The *head term* $\Upsilon_{<_M}(P)$ of a polynomial P with respect to a monomial order $<_M$ is the term of P whose corresponding monomial is greatest under $<_M$. The polynomial Q in \mathbb{X} is called *reducible* by the polynomial P in \mathbb{X} with respect to $<_M$ when Q contains a term whose corresponding monomial $V = W \cdot \Upsilon_{<_M}(P)$ for some monomial W in \mathbb{X} . In this case, the polynomial $R = Q - W \cdot P$ is called the *reduction* of Q by P , denoted $Q \xrightarrow{P} R$. In what follows, we may omit the monomial order with respect to which reductions are specified, with the understanding that a consistent monomial order is assumed.

For a set \mathbb{M} of polynomials in \mathbb{X} , Q is called *reducible* by \mathbb{M} when Q is reducible by some polynomial in \mathbb{M} , $\exists P \in \mathbb{M} Q \xrightarrow{P} R$, and we write $Q \xrightarrow{\mathbb{M}} R$. We write $Q \xrightarrow{\mathbb{M}}^* R$ when Q can be reduced to R by a finite number of reduction steps, $Q \xrightarrow{\mathbb{M}} \dots \xrightarrow{\mathbb{M}} R$. We write $Q \xrightarrow{\mathbb{M}}^* R$ when $Q \xrightarrow{\mathbb{M}} R$ and R is not reducible by \mathbb{M} , in which case R is called an \mathbb{M} -normal form of Q .

A *Gröbner basis* in \mathbb{X} over \mathbb{K} is a finite set of polynomials $\mathbb{G} \subseteq \mathbb{K}[\mathbb{X}]$, $|\mathbb{G}| < \infty$, for which the \mathbb{G} -normal form of every polynomial in \mathbb{X} is unique, $\forall P \in \mathbb{K}[\mathbb{X}] P \xrightarrow{\mathbb{G}}^* Q \wedge P \xrightarrow{\mathbb{G}}^* R \Rightarrow Q = R$. The Gröbner basis \mathbb{G} with respect to the lexicographic order $<_L$ has the property that the subset of the ideal generated by \mathbb{G} containing only polynomials in a subset $\mathbb{X}_p = \{x_i\}_{i \in \langle p \rangle} \subseteq \mathbb{X}$ of the variables \mathbb{X} , called its *pth elimination ideal* $\Psi_p(\mathbb{G}) \equiv \Psi(\mathbb{G}) \cap \mathbb{K}[\mathbb{X}_p]$, is equal to the ideal generated by the subset of \mathbb{G} containing only \mathbb{X}_p , $\Psi_p(\mathbb{G}) = \Psi(\mathbb{G} \cap \mathbb{K}[\mathbb{X}_p])$. In other words, $\mathbb{G} \cap \mathbb{K}[\mathbb{X}_p]$ is a basis for $\Psi_p(\mathbb{G})$.

Given a set \mathbb{M} of polynomials in \mathbb{X} over \mathbb{K} , there exist algorithms to compute a Gröbner basis \mathbb{G} with respect to $<_L$ satisfying $\Psi(\mathbb{G}) = \Psi(\mathbb{M})$ [6]. When the polynomials \mathbb{M} have finitely many common roots over \mathbb{K} , the set \mathbb{G} can be used to calculate these roots as follows: The roots of x_i in \mathbb{M} are the roots of $\Psi_i(\mathbb{M}) = \Psi_i(\mathbb{G})$, which are the roots of the basis $\mathbb{G} \cap \mathbb{K}[\mathbb{X}_i]$. The basis $\mathbb{G} \cap \mathbb{K}[\mathbb{X}_1]$ consists of univariate polynomials in x_1 , whose roots can be calculated by standard methods, and substituted into $\mathbb{G} \cap \mathbb{K}[\mathbb{X}_2]$ to obtain sets of univariate polynomials in x_2 , whose roots are substituted into $\mathbb{G} \cap \mathbb{K}[\mathbb{X}_3]$ and so on, yielding the complete set of roots of \mathbb{M} in \mathbb{X} .

2.2 Graph Theory

Graph theory is used throughout the text to formalize the structural relationships between elements of MDPL models in Chapters 3 and 4, and to describe algorithmic components of the partial inversion

transformation in §4.3.

2.2.1 Graphs and Digraphs

A (simple) *undirected graph* or *graph* is a tuple $\mathfrak{G} = [\mathbb{V}, \mathbf{E}]$ on *vertices* \mathbb{V} with *edges* \mathbf{E} where each edge joins two distinct vertices, $\mathbf{E} \subseteq \mathcal{P}_2(\mathbb{V})$. Similarly, a (simple) *directed graph* or *digraph* is a tuple $\mathfrak{H} = [\mathbb{V}, \mathcal{E}]$ on vertices \mathbb{V} with edges \mathcal{E} where each edge joins two distinct vertices in a particular order, $\mathcal{E} \subseteq \mathbb{V}^\otimes$, so that each edge has a direction. Herein we tacitly assume any (di)graph to be finite, $|\mathbb{V}| < \infty$, unless otherwise specified.

In a graph $\mathfrak{G} = [\mathbb{V}, \mathbf{E}]$ or digraph $\mathfrak{H} = [\mathbb{V}, \mathcal{E}]$, a vertex $v \in \mathbb{V}$ or $v \in \mathcal{E}$ joined by an edge $\mathbf{E} \in \mathbf{E}$ or $\mathcal{E} \in \mathcal{E}$ is called *incident* on \mathbf{E} or \mathcal{E} , respectively. Similarly, the edges $\{\mathbf{E} \in \mathbf{E} \mid v \in \mathbf{E}\}$ or $\{\mathcal{E} \in \mathcal{E} \mid v \in \mathcal{E}\}$ joining a vertex $v \in \mathbb{V}$ are called incident on v . These relationships are commonly represented diagrammatically, where a vertex $v \in \mathbb{V}$ appears as a point (\bullet), an undirected edge $\{v, w\} \in \mathbf{E}$ appears as a line ($—$) connecting v and w , and a directed edge $[v, w] \in \mathcal{E}$ appears as an arrow (\rightarrow) from v to w .

Two graphs $\mathfrak{G} = [\mathbb{V}, \mathbf{E}]$ and $\mathfrak{G}' = [\mathbb{V}', \mathbf{E}']$ are called *isomorphic*, $\mathfrak{G} \cong \mathfrak{G}'$, when there is a bijective function $g : \mathbb{V} \longleftrightarrow \mathbb{V}'$ such that the connections in each graph between corresponding vertices under g are identical, $\mathbf{E}' = \{g(\mathbf{E})\}_{\mathbf{E} \in \mathbf{E}}$. Similarly, two digraphs $\mathfrak{H} = [\mathbb{V}, \mathcal{E}]$ and $\mathfrak{H}' = [\mathbb{V}', \mathcal{E}']$ are called isomorphic, $\mathfrak{H} \cong \mathfrak{H}'$, when there exists $h : \mathbb{V} \longleftrightarrow \mathbb{V}'$ satisfying $\mathcal{E}' = \{h(\mathcal{E})\}_{\mathcal{E} \in \mathcal{E}}$.

A *subgraph* of a graph $\mathfrak{G} = [\mathbb{V}, \mathbf{E}]$ is a graph $\mathfrak{G}' = [\mathbb{V}', \mathbf{E}']$ containing a subset of its edges $\mathbf{E}' \subseteq \mathbf{E}$. When \mathfrak{G}' contains all edges in \mathfrak{G} incident on its vertices, $\mathbf{E}' = \mathcal{P}_2(\mathbb{V}') \cap \mathbf{E}$, we call \mathfrak{G}' the subgraph of \mathfrak{G} (*vertex-*)*induced* by \mathbb{V}' , denoted $\mathfrak{G}[\mathbb{V}']$. When \mathfrak{G}' contains only vertices incident on its edges, $\mathbb{V}' = \bigcup \mathbf{E}'$, we call it the subgraph (*edge-*)*induced* by \mathbf{E}' , denoted $\mathfrak{G}[\mathbf{E}']$. Similarly, a subgraph of a digraph $\mathfrak{H} = [\mathbb{V}, \mathcal{E}]$ is a digraph $\mathfrak{H}' = [\mathbb{V}', \mathcal{E}']$ where $\mathcal{E}' \subseteq \mathcal{E}$, which is called the vertex-induced subgraph $\mathfrak{H}[\mathbb{V}']$ when $\mathcal{E}' = \mathbb{V}'^\otimes \cap \mathcal{E}$ and the edge-induced subgraph $\mathfrak{H}[\mathcal{E}']$ when $\mathbb{V}' = \bigcup_{\mathcal{E} \in \mathcal{E}'} \langle \mathcal{E} \rangle$.

The *disorientation* of a digraph $\mathfrak{H} = [\mathbb{V}, \mathcal{E}]$ is the undirected graph $\overline{\mathfrak{H}} \equiv [\mathbb{V}, \overline{\mathcal{E}}]$ where the directed edges in \mathcal{E} are replaced by their corresponding undirected edges in $\overline{\mathcal{E}} \equiv \{\langle \mathcal{E} \rangle\}_{\mathcal{E} \in \mathcal{E}}$. In contrast, an *orientation* of an undirected graph $\mathfrak{G} = [\mathbb{V}, \mathbf{E}]$ is a directed graph $\overrightarrow{\mathfrak{G}} = [\mathbb{V}, \overrightarrow{\mathbf{E}}]$ where each edge in \mathbf{E} is given a single direction in $\overrightarrow{\mathbf{E}}$, so that $\overrightarrow{\overrightarrow{\mathbf{E}}} = \mathbf{E}$ and $|\overrightarrow{\mathbf{E}}| = |\mathbf{E}|$. We have $\overrightarrow{\overrightarrow{\mathfrak{G}}} = \mathfrak{G}$ for any orientation $\overrightarrow{\mathfrak{G}}$ of \mathfrak{G} .

2.2.2 Paths and Cycles

In a graph $\mathfrak{G} = [\mathbb{V}, \mathbf{E}]$, a vertex $w \in \mathbb{V}$ is called a *neighbor* of (or *adjacent* to) another vertex $v \in \mathbb{V}$ when there is an edge $\{v, w\} \in \mathbf{E}$ incident on both. The set of neighbors of v is called the *neighborhood* of v , $N_{\mathfrak{G}}(v) \equiv \{w \in \mathbb{V} \mid \{v, w\} \in \mathbf{E}\}$. The neighborhood of a subset of vertices $\mathbb{U} \subseteq \mathbb{V}$ includes all vertices in the neighborhoods of the vertices in \mathbb{U} , $N_{\mathfrak{G}}(\mathbb{U}) \equiv \bigcup_{u \in \mathbb{U}} N_{\mathfrak{G}}(u)$. Similarly, a

k -neighbor of v is a vertex in the k -neighborhood of v defined by $N_{\mathfrak{G}}^k(v) \equiv \bigcup_{i \in \langle k \rangle} N_{\mathfrak{G}}^k(v)$. We define $N_{0\mathfrak{G}}^k(v) \equiv \{v\} \cup N_{\mathfrak{G}}^k(v)$.

In a digraph $\mathfrak{H} = [\mathbb{V}, \mathcal{E}]$, a vertex $w \in \mathbb{V}$ is called a *parent* of $v \in \mathbb{V}$ when there is an edge $[w, v] \in \mathcal{E}$ from w to v , and a *child* of v when there is an edge $[v, w] \in \mathcal{E}$ from v to w . The set of parents of v is called the *in-neighborhood* of v , $N_{\mathfrak{H}}^-(v) \equiv \{w \in \mathbb{V} \mid [w, v] \in \mathcal{E}\}$, and the set of children of v is called the *out-neighborhood* of v , $N_{\mathfrak{H}}^+(v) \equiv \{w \in \mathbb{V} \mid [v, w] \in \mathcal{E}\}$. Either a parent or child of v is called a neighbor of v . Hence the neighborhood in a digraph includes both the in-neighborhood and out-neighborhood, $N_{\mathfrak{H}}(v) \equiv N_{\mathfrak{H}}^-(v) \cup N_{\mathfrak{H}}^+(v) \equiv N_{\overline{\mathfrak{H}}}(v)$. The k -neighbors and -neighborhoods are defined as above for undirected graphs, and likewise the k -in-neighborhood $N_{\mathfrak{H}}^{k-}(v) \equiv \bigcup_{i \in \langle k \rangle} N_{\mathfrak{G}}^{-k}(v)$ and k -out-neighborhood $N_{\mathfrak{H}}^{k+}(v) \equiv \bigcup_{i \in \langle k \rangle} N_{\mathfrak{G}}^{+k}(v)$. We define $N_{0\mathfrak{G}}^{k+}(v) \equiv \{v\} \cup N_{\mathfrak{G}}^{k+}(v)$ and $N_{0\mathfrak{G}}^{k-}(v) \equiv \{v\} \cup N_{\mathfrak{G}}^{k-}(v)$. A vertex $w \in N_{\mathfrak{H}}^{\infty-}(v)$ is called a *predecessor* of v , while a vertex $w \in N_{\mathfrak{H}}^{\infty+}(v)$ is called a *successor* of v .

In the graph \mathfrak{G} , the size of the neighborhood of the vertex v is called the *degree* of v , $|v|_{\mathfrak{G}} \equiv |N_{\mathfrak{G}}(v)|$. Likewise, the degree of a subset $\mathbb{U} \subseteq \mathbb{V}$ of vertices is $|\mathbb{U}|_{\mathfrak{G}} \equiv |N_{\mathfrak{G}}(\mathbb{U})|$. Similarly, in the digraph \mathfrak{H} , the sizes of the in-neighborhood and out-neighborhood of the vertex v are called the *indegree* and *outdegree*, $|v|_{\mathfrak{H}}^- \equiv |N_{\mathfrak{H}}^-(v)|$ and $|v|_{\mathfrak{H}}^+ \equiv |N_{\mathfrak{H}}^+(v)|$, respectively, and likewise $|\mathbb{U}|_{\mathfrak{H}}^- \equiv |N_{\mathfrak{H}}^-(\mathbb{U})|$ and $|\mathbb{U}|_{\mathfrak{H}}^+ \equiv |N_{\mathfrak{H}}^+(\mathbb{U})|$ for $\mathbb{U} \subseteq \mathbb{V}$.

In the graph \mathfrak{G} , given a list $\mathcal{P} \subseteq \mathbb{V}$ of $n+1$ vertices, $|\mathcal{P}| = n+1$, $n \in \mathbb{N}$, consecutively connected by edges, $\forall_{i \in \langle n \rangle} \mathcal{P}_{i+1} \in N_{\mathfrak{G}}(\mathcal{P}_i)$, the set of connecting edges $\mathbf{P} = \{\{\mathcal{P}_i, \mathcal{P}_{i+1}\}\}_{i \in \langle n \rangle}$, is called a *walk* of length n from \mathcal{P}_1 to \mathcal{P}_{n+1} , and the vertices \mathcal{P} are called the *steps* of \mathbf{P} . Similarly, in the digraph \mathfrak{H} , given $\forall_{i \in \langle n \rangle} \mathcal{P}_{i+1} \in N_{\mathfrak{H}}^+(\mathcal{P}_i)$, the set of connecting edges $\mathcal{P} = \{\{\mathcal{P}_i, \mathcal{P}_{i+1}\}\}_{i \in \langle n \rangle}$, is called a walk of length n from \mathcal{P}_1 to \mathcal{P}_{n+1} , and the vertices \mathcal{P} are called the steps of \mathcal{P} . When $\mathcal{P}_{n+1} = \mathcal{P}_1$, the walk \mathbf{P} is called a *tour*. If the vertices \mathcal{P} are all distinct, $\mathcal{P} \subseteq (\mathcal{P})$, the walk \mathbf{P} or \mathcal{P} is called a *path*. If the vertices \mathcal{P} are distinct except that $\mathcal{P}_{n+1} = \mathcal{P}_1$, the walk \mathbf{P} or \mathcal{P} is called a *cycle* of length n . The k -neighborhood and k -out-neighborhood of $v \in \mathbb{V}$ are the sets of vertices to which there is a walk of length k , or a path of length $n \leq k$, from v in \mathfrak{G} and \mathfrak{H} , respectively.

The graph \mathfrak{G} is called *connected* when there is a path from a vertex v to any other vertex, $\exists_{v \in \mathbb{V}} N_{0\mathfrak{G}}^{\infty}(v) = \mathbb{V}$, or equivalently, from any vertex to any other vertex, $\forall_{v \in \mathbb{V}} N_{0\mathfrak{G}}^{\infty}(v) = \mathbb{V}$. The *connected component* of the vertex v is the connected subgraph $\mathfrak{G}[N_{0\mathfrak{G}}^{\infty}(v)]$ of \mathfrak{G} induced by v and the vertices to which there is a path from v . The subgraphs $\{\mathfrak{G}[N_{0\mathfrak{G}}^{\infty}(v)]\}_{v \in \mathbb{V}}$ that are connected components of some vertex in \mathfrak{G} are called the connected components of \mathfrak{G} . Hence \mathfrak{G} is connected when it is the only connected component of itself.

The digraph \mathfrak{H} is called *weakly connected* when $\overline{\mathfrak{H}}$ is connected. The *weakly connected components* of \mathfrak{H} are the connected components of $\overline{\mathfrak{H}}$. The digraph \mathfrak{H} is called *strongly connected* when there is a path from any vertex to any other vertex, $\forall_{v \in \mathbb{V}} N_{0\mathfrak{H}}^{\infty+}(v) = \mathbb{V}$. The *strongly connected component* of the vertex v is the strongly connected subgraph $\mathfrak{H}[N_{0\mathfrak{H}}^{\infty-}(v) \cap N_{0\mathfrak{H}}^{\infty+}(v)]$ of \mathfrak{H} induced by the vertices

for which there is a path to and from v . The subgraphs $\{\mathfrak{H}[N_{0\mathfrak{H}}^{\infty-}(v) \cap N_{0\mathfrak{H}}^{\infty+}(v)]\}_{v \in \mathbb{V}}$ that are strongly connected components of some vertex in \mathfrak{H} are called the strongly connected components of \mathfrak{H} . Hence \mathfrak{H} is strongly connected when it is the only strongly connected component of itself.

For the digraph \mathfrak{H} , we define $v \prec_{\mathfrak{H}} w \equiv w \succ_{\mathfrak{H}} v \equiv w \in N_{\mathfrak{H}}^{\infty+}(v) \equiv v \in N_{\mathfrak{H}}^{\infty-}(w)$ for any two vertices $v, w \in \mathbb{V}$, and say that v precedes w while w succeeds v . We write $v \prec_{\mathfrak{H}} w$ to indicate that either $v = w$ or $v \prec_{\mathfrak{H}} w$.

A digraph with no cycles is called an *acyclic digraph* or *directed acyclic graph* (DAG). When \mathfrak{H} is a DAG, a vertex w cannot both precede and succeed another vertex v , since this would imply that \mathfrak{H} contains both a path from w to v and a path from v to w , and hence at least one cycle. Hence the relation $\prec_{\mathfrak{H}}$ is a partial order on \mathbb{V} , where $v \prec_{\mathfrak{H}} w \Rightarrow v \not\succeq_{\mathfrak{H}} w$. In other words, each vertex is in its own strongly connected component.

A graph with no cycles is called a *forest*. A connected forest is called a *tree*, in which there is exactly one path from any vertex to any other vertex. A weakly connected (acyclic) digraph \mathfrak{H} for which $\overline{\mathfrak{H}}$ is a tree is called a *directed tree* (or *rooted tree*) when there is a vertex $u \in \mathbb{V}$ called the *root* that precedes all others, $\forall_{v \in \mathbb{V}} u \prec_{\mathfrak{H}} v$. When \mathfrak{G} is a tree, for each vertex $u \in \mathbb{V}$ there is a unique orientation $\overrightarrow{\mathfrak{G}}$ that is a directed tree with u as the root.

2.2.3 Partite Graphs and Matrices

The digraph $\mathfrak{H} = [\mathbb{V}, \mathcal{E}]$ or graph $\overline{\mathfrak{H}}$ is called *bipartite* on $\{\mathbb{X}, \mathbb{Y}\}$ when its vertices can be partitioned into two disjoint sets \mathbb{X}, \mathbb{Y} , $\mathbb{V} = \mathbb{X} \sqcup \mathbb{Y}$, so that each edge joins one vertex in \mathbb{X} and one vertex in \mathbb{Y} , $\mathcal{E} \subseteq \mathbb{X} \times \mathbb{Y} \sqcup \mathbb{Y} \times \mathbb{X}$. Equivalently, no two vertices in \mathbb{X} or two vertices in \mathbb{Y} are adjacent, $\mathbb{X} \cap N_{\mathfrak{H}}(\mathbb{X}) = \mathbb{Y} \cap N_{\mathfrak{H}}(\mathbb{Y}) = \emptyset$. More generally, \mathfrak{H} or $\overline{\mathfrak{H}}$ is called *n-partite* on \mathbb{X} for $n \in \mathbb{N}$ when its vertices can be partitioned into n disjoint sets $\mathbb{X} = \{\mathbb{X}_i\}_{i \in \langle n \rangle}$, $\mathbb{V} = \bigsqcup \mathbb{X}$, so that each edge joins two vertices in different sets, $\mathcal{E} \subseteq \bigsqcup_{\mathbb{X} \in \mathbb{X}^{\otimes}} \prod \mathbb{X}$. Equivalently, no two vertices in a single \mathbb{X}_i are adjacent, $\forall_{i \in \langle n \rangle} \mathbb{X}_i \cap N_{\mathfrak{H}}(\mathbb{X}_i) = \emptyset$. A 3-partite (di)graph is called *tripartite*.

A *bigraph* is a tuple $\mathfrak{G} = [\mathbb{X}, \mathbb{Y}, \mathbf{E}]$ such that $[\mathbb{X} \sqcup \mathbb{Y}, \mathbf{E}]$ is a bipartite graph on $\{\mathbb{X}, \mathbb{Y}\}$. Here the elements of \mathbb{X} and \mathbb{Y} are called vertices of \mathfrak{G} , the elements of \mathbf{E} are called edges of \mathfrak{G} , and other definitions of graph theory are analogous. A vertex $x \in \mathbb{X}$ is called a *row vertex* of \mathfrak{G} , while a vertex $y \in \mathbb{Y}$ is called a *column vertex* of \mathfrak{G} . An *ordering* of \mathfrak{G} is an *ordered bigraph* $[\mathcal{X}, \mathcal{Y}, \mathbf{E}]$ where $\mathcal{X} \subseteq \mathbb{X}$ and $\mathcal{Y} \subseteq \mathbb{Y}$. All orderings of \mathfrak{G} are isomorphic to one another. Each ordering $[\mathcal{X}, \mathcal{Y}, \mathbf{E}]$ of the bigraph \mathfrak{G} has a corresponding binary matrix $\mathbf{A} \in \{0, 1\}^{m \times n}$ in which the row vertices of \mathfrak{G} correspond to rows of \mathbf{A} , $m = |\mathcal{X}|$, the column vertices of \mathfrak{G} correspond to columns of \mathbf{A} , $n = |\mathcal{Y}|$, and the edges of \mathfrak{G} correspond to the nonzero entries of \mathbf{A} , $\forall_{i \in \langle m \rangle} \forall_{j \in \langle n \rangle} [\mathcal{X}_i, \mathcal{Y}_j] \in \mathbf{E} \Leftrightarrow A_{ij} \neq 0$.

A *p-permutation* is a function $\sigma : \langle\langle p \rangle\rangle \longleftrightarrow \langle\langle p \rangle\rangle$ reordering the integers $1, 2, \dots, p$. The set of all *p-permutations* is denoted \mathbb{S}_p . A *permutation* of a *p-tuple* \mathcal{V} , $p = |\mathcal{V}|$, is a tuple $\mathcal{V}_{\sigma(\langle p \rangle)}$ rearranging its elements according to a *p-permutation* $\sigma \in \mathbb{S}_p$. A permutation $\mathbf{A}_{\sigma(\langle m \rangle)}$ of a matrix $\mathbf{A}_{m \times n}$ for

$\sigma \in \mathbb{S}_m$ is called a *row permutation* of A , while a matrix $((A^T)_{\sigma((n))})^T$ for $\sigma \in \mathbb{S}_n$ is called a *column permutation* of A . A *row-column permutation* of A is a row permutation of a column permutation of A , or equivalently, a column permutation of a row permutation of A . All matrices of orderings of a bigraph \mathcal{G} are row-column permutations of one another.

2.2.4 Bigraph Decomposition

In a bigraph $\mathcal{G} = [\mathbb{X}, \mathbb{Y}, \mathbb{E}]$, a *matching* is a subset $\mathbb{M} \subseteq \mathbb{E}$ of edges such that no two edges in \mathbb{M} are incident on the same vertex, $\sqcup \mathbb{M}$. A vertex $v \in \sqcup \mathbb{M}$ incident on an edge in \mathbb{M} is called *matched* under \mathbb{M} , otherwise $v \notin \sqcup \mathbb{M}$ is called *unmatched* under \mathbb{M} . The matching \mathbb{M} is called *maximum* in \mathcal{G} when no matching in \mathcal{G} has more edges, $\forall \mathbb{N} \in \mathcal{P}(\mathbb{E}) \sqcup \mathbb{N} \ |\mathbb{N}| \leq |\mathbb{M}|$. The matching \mathbb{M} is called *row-perfect* when every row vertex of \mathcal{G} is matched, $\mathbb{X} \subseteq \sqcup \mathbb{M}$, *column-perfect* when every column vertex of \mathcal{G} is matched, $\mathbb{Y} \subseteq \sqcup \mathbb{M}$, and *perfect* when every vertex is matched, $\mathbb{X} \sqcup \mathbb{Y} \subseteq \sqcup \mathbb{M}$.

A binary matrix A corresponding to an ordering of \mathcal{G} is said to have a perfect, row-perfect, or column-perfect matching whenever \mathcal{G} has a perfect, row-perfect, or column-perfect matching, respectively. A matrix A is called *underdetermined* or *horizontal* when it has fewer rows than columns, $|A| < |A^T|$, *well determined* or *square* when it has the same number of rows and columns, $|A| = |A^T|$, and *overdetermined* or *vertical* when it has more rows than columns, $|A| > |A^T|$. A square matrix A is called *structurally nonsingular* when it has a perfect matching, and *structurally singular* when it does not.

An *alternating path* in \mathcal{G} with respect to \mathbb{M} is a path \mathbb{P} in \mathcal{G} from $v \in \mathbb{X} \sqcup \mathbb{Y}$ to $w \in \mathbb{X} \sqcup \mathbb{Y}$ in which alternating edges along \mathbb{P} are in \mathbb{M} , $\sqcup \mathbb{P} \setminus \{v, w\} \subseteq \sqcup \mathbb{P} \cap \mathbb{M}$. Similarly, an *alternating cycle* or *alternating tour* in \mathcal{G} with respect to \mathbb{M} is a cycle or tour \mathbb{Q} in \mathcal{G} , respectively, satisfying $\sqcup \mathbb{Q} = \sqcup \mathbb{Q} \cap \mathbb{M}$. When the alternating path \mathbb{P} begins and ends with unmatched vertices, $\sqcup \mathbb{P} \setminus \{v, w\} = \sqcup \mathbb{P} \cap \mathbb{M}$, it is called an *augmenting path* in \mathcal{G} with respect to \mathbb{M} .

Berge [5] showed that \mathbb{M} is maximum in \mathcal{G} if and only if there is no augmenting path in \mathcal{G} with respect to \mathbb{M} . The best known algorithms for finding maximum matchings in graphs repeatedly search for an augmenting path \mathbb{P} with respect to a known matching \mathbb{M} and compute a larger matching $\mathbb{M}' = \mathbb{M} \oplus \mathbb{P}$ with $|\mathbb{M}'| = |\mathbb{M}| + 1$. Galil [28] gives a detailed discussion of these algorithms. For general bigraphs, the lowest worst-case complexity bound known is that of the Hopcroft-Karp algorithm [33].

The *Dulmage-Mendelsohn decomposition* [22, 36] of \mathcal{G} with corresponding binary matrix A and a maximum matching \mathbb{M} consists of row-subsets $[\mathbb{X}_h, \mathbb{X}_s, \mathbb{X}_v]$ of \mathbb{X} and column-subsets $[\mathbb{Y}_h, \mathbb{Y}_s, \mathbb{Y}_v]$ of \mathbb{Y} , where: $\mathbb{X}_h \sqcup \mathbb{Y}_h$ and $\mathbb{X}_v \sqcup \mathbb{Y}_v$ contain those vertices $w \in \mathbb{X} \sqcup \mathbb{Y}$ equal to, or to which there is an alternating path in \mathcal{G} with respect to \mathbb{M} from, an unmatched row vertex $v \in \mathbb{X} \setminus \sqcup \mathbb{M}$ or column vertex $v \in \mathbb{Y} \setminus \sqcup \mathbb{M}$, respectively; and $\mathbb{X}_s = \mathbb{X} \setminus (\mathbb{X}_h \cup \mathbb{X}_v)$ and $\mathbb{Y}_s = \mathbb{Y} \setminus (\mathbb{Y}_h \cup \mathbb{Y}_v)$ contain the remaining row and column vertices, respectively. We define $A_{xy} \equiv A_{\mathbb{X}_x \mathbb{Y}_y}$ for $x, y \in \{h, s, v\}$ and

$A_{hsv} = A_{\mathcal{X}\mathcal{Y}}$ for $\mathcal{X} \subseteq [\mathbb{X}_h, \mathbb{X}_s, \mathbb{X}_v]$ and $\mathcal{Y} \subseteq [\mathbb{Y}_h, \mathbb{Y}_s, \mathbb{Y}_v]$. The Dulmage-Mendelsohn decomposition has the following properties [54]:

1. The row- and column-subsets of the decomposition are pairwise disjoint, $\mathbb{X} = \mathbb{X}_h \sqcup \mathbb{X}_s \sqcup \mathbb{X}_v$ and $\mathbb{Y} = \mathbb{Y}_h \sqcup \mathbb{Y}_s \sqcup \mathbb{Y}_v$.
2. The submatrices A_{hh} , A_{ss} , and A_{vv} are horizontal, square, and vertical, respectively, $|\mathbb{X}_h| < |\mathbb{Y}_h|$, $|\mathbb{X}_s| = |\mathbb{Y}_s|$, $|\mathbb{X}_v| > |\mathbb{Y}_v|$.
3. The matching \mathbf{M} matches only vertices in corresponding row- and column-subsets, $\mathbf{M} \subseteq \bigsqcup_{z \in \{h,s,v\}} (\mathbb{X}_z \times \mathbb{Y}_z)$.
4. The matching \mathbf{M} perfectly matches the row vertices \mathbb{X}_s to the column vertices \mathbb{Y}_s , $\mathbb{X}_s \sqcup \mathbb{Y}_s \subseteq \bigsqcup \mathbf{M}$.
5. The edges \mathbf{E} do not join any row vertices \mathbb{X}_s to column vertices \mathbb{Y}_h or row vertices \mathbb{X}_v to column vertices $\mathbb{Y}_h \sqcup \mathbb{Y}_s$,

$$A_{hsv} = \begin{pmatrix} A_{hh} & A_{hs} & A_{hv} \\ 0 & A_{ss} & A_{sv} \\ 0 & 0 & A_{vv} \end{pmatrix}.$$

6. The decomposition is identical for any maximum matching \mathbf{M} of \mathcal{G} .

Given the last property, the Dulmage-Mendelsohn decomposition of \mathcal{G} is well defined without specifying the matching \mathbf{M} . The row- and column-subsets of this decomposition can be further decomposed to yield a *block triangular decomposition* of \mathcal{G} consisting of row-subsets $[\mathbb{X}_{zi}]_{i \in \langle p_z \rangle}$ and column-subsets $[\mathbb{Y}_{zi}]_{i \in \langle p_z \rangle}$ for $z \in \{h, s, v\}$ such that:

1. The induced subgraphs $\{\mathcal{G}[\mathbb{X}_{zi} \sqcup \mathbb{Y}_{zi}]_{i \in \langle p_z \rangle}\}$ are the connected components of $\mathcal{G}[\mathbb{X}_z \sqcup \mathbb{Y}_z]$ for $z \in \{h, v\}$.
2. The row-subsets $\{\mathbb{X}_{si}\}_{i \in \langle p_s \rangle}$ are the equivalence classes of the equivalence relation \sim on \mathbb{X}_s such that $v \sim w$ for two row vertices $v, w \in \mathbb{X}_s$ whenever there is an alternating tour \mathbf{P} in \mathcal{G} with respect to \mathbf{M} containing them, $v, w \in \bigsqcup \mathbf{P}$.
3. The column-subsets $\{\mathbb{Y}_{si}\}_{i \in \langle p_s \rangle}$ consist of the column vertices matched to the corresponding row-subsets, $\forall_{i \in \langle p_s \rangle} \mathbb{Y}_{si} = \mathbb{Y}_s \cap \{\mathbf{E} \in \mathbf{M} \mid \mathbf{E} \cap \mathbb{X}_{si} \neq \emptyset\}$.
4. The row-subsets and column-subsets $\{\mathbb{X}_{si}, \mathbb{Y}_{si}\}_{i \in \langle p_s \rangle}$ are ordered so that $A_{\mathcal{X}_s \mathcal{Y}_s}$ is in block upper triangular form, where $\mathcal{X}_s \subseteq [\mathbb{X}_{si}]_{i \in \langle p_s \rangle}$ and $\mathcal{Y}_s \subseteq [\mathbb{Y}_{si}]_{i \in \langle p_s \rangle}$, $\forall_{i \in \langle p_s \rangle} \forall_{j \in \langle p_s \rangle} (\mathbb{X}_{si} \times \mathbb{Y}_{sj}) \cap \mathbf{E} \neq \emptyset \Rightarrow i \leq j$.

Like the Dulmage-Mendelsohn decomposition, a block triangular decomposition of \mathcal{G} is independent of the choice of \mathbf{M} . However, the ordering of row-subsets $[\mathbb{X}_{zi}]_{i \in \langle p_z \rangle}$ and column-subsets $[\mathbb{Y}_{zi}]_{i \in \langle p_z \rangle}$ is arbitrary for $z \in \{h, v\}$. The row-subsets $\{\mathbb{X}_{si}\}_{i \in \langle p_s \rangle}$ are the vertex sets of the strongly connected components of the digraph $\mathfrak{H}_s = [\mathbb{X}_s, \mathcal{E}_s]$ defined by orienting the edges of $\mathcal{G}[\mathbb{X}_s \sqcup \mathbb{Y}_s]$ from column vertices to row vertices and then merging the column vertices into the corresponding row vertices to which they are matched, $\mathcal{E}_s = \{[u, w] \in \mathbb{X}_s^\otimes \mid \exists v \in \mathbb{Y}_s \{u, v\} \in \mathbf{M} \wedge \{v, w\} \in \mathbf{E}\}$ [54]. Hence a block triangular decomposition can be found from the Dulmage-Mendelsohn decomposition by finding the connected components of $\mathcal{G}[\mathbb{X}_z \sqcup \mathbb{Y}_z]$ for $z \in \{h, v\}$, finding the strongly connected components of \mathfrak{H}_s , and ordering the latter in block upper triangular form.

2.3 Theory of Computation

Some key ideas in the theory of computation are used to formalize the arguments presented in §3.4 on the computational expressivity of MDPL. Much of the terminology herein is nonstandard, but provides a clarification of otherwise ambiguous notions.

2.3.1 Grammars

In computer science, a *string* is a linear arrangement of distinguishable symbols. For example, a string consisting of the symbol a followed by the symbol b is written ab . The symbol ϵ denotes the *empty string*, a string containing no symbols. Where applicable, a symbol may be considered a string consisting of a single symbol.

The *concatenation* ST of two strings S and T is the string containing the symbols of one followed by the symbols of the other. Concatenation can be applied to multiple strings. For example, the concatenation of the string S , the string ab , and the symbol c is the string $Sabc$. Concatenation can also be applied over sets of strings. For example, the concatenation of the set of symbols $\{a, b\}$ with the string S is the set of strings $\{aS, bS\}$. Given a set of strings \mathbb{V} , defining $\mathbb{V}_0 = \{\epsilon\}$ and $\mathbb{V}_{i+1} = \mathbb{V}_i \mathbb{V}$ for $i \in \mathbb{Z}^*$, the set of all strings that can be constructed by concatenating strings in \mathbb{V} is the *Kleene closure* of \mathbb{V} defined by $\mathbb{V}^* \equiv \bigcup_{i \in \mathbb{Z}^*} \mathbb{V}_i$.

A *grammar* [10] is a tuple $[\mathbb{S}, \mathbb{T}, \mathbb{U}, s]$ where \mathbb{S} is a finite set of symbols called *nonterminals*, \mathbb{T} is a finite set of symbols called *terminals*, \mathbb{S} and \mathbb{T} are disjoint ($\mathbb{S} \sqcup \mathbb{T}$), $s \in \mathbb{S}$ is a particular nonterminal called the *start symbol*, and \mathbb{U} is a finite set of *production rules*, where each production rule $U \mapsto V$ relates a string of (non)terminals $U \in (\mathbb{S} \sqcup \mathbb{T})^* \mathbb{S} (\mathbb{S} \sqcup \mathbb{T})^*$ containing at least one nonterminal to another string of (non)terminals $V \in (\mathbb{S} \sqcup \mathbb{T})^*$. The notation $U \mapsto \{V, W, \dots\}$ or $U \mapsto V \mid W \mid \dots$ indicates several production rules $U \mapsto V$, $U \mapsto W$, \dots

Example 2.3.1. $\hat{\mathcal{G}} = [\{s, t\}, \{a, b, c, \triangleright\}, \{s \mapsto t \mid s \triangleright s, t \mapsto a \mid b \mid c\}, s]$ is a grammar.

An *application* of a production rule $U \mapsto V$ to a string SUT is the string SVT , where S and T are strings. A *production* in a grammar $\mathcal{G} = [\mathbb{S}, \mathbb{T}, \mathbb{U}, s]$ is defined recursively as either the start symbol s of \mathcal{G} or any application of a production rule $u \in \mathbb{U}$ to a production in \mathcal{G} . A *statement* in \mathcal{G} is a production consisting only of terminals in \mathcal{G} . The set of all statements in \mathcal{G} is called the *language* of \mathcal{G} , denoted $\Lambda(\mathcal{G})$.

Example 2.3.2. The strings a , $b \triangleright a$, and $a \triangleright c \triangleright a$ are statements in $\hat{\mathcal{G}}$, hence $\{a, b \triangleright a, a \triangleright c \triangleright a\} \subseteq \Lambda(\hat{\mathcal{G}})$.

2.3.2 Translators

A *pattern* in a grammar \mathcal{G} is a string consisting of terminals and nonterminals of \mathcal{G} together with a finite number of symbols called *schematic variables*. An *instantiation* of patterns S, T, \dots in \mathcal{G} is the corresponding strings S', T', \dots that result when all instances of each distinct schematic variable in S, T, \dots are replaced by an identical statement in \mathcal{G} . A *statement pattern* in \mathcal{G} is a pattern S such that any instantiation of S is a statement in \mathcal{G} .

A *translation rule* $V \leftrightarrow W$ between a grammar \mathcal{G} and a grammar \mathcal{H} consists of a statement pattern V in \mathcal{G} and a statement pattern W in \mathcal{H} containing the same schematic variables. For any instantiation V', W' of V, W in both \mathcal{G} and \mathcal{H} , any application of the production rule $V' \mapsto W'$ to a string S is called an application of $V \leftrightarrow W$ to S . As with production rules, we indicate multiple translation rules $U \leftrightarrow V, U \leftrightarrow W, \dots$ by $U \leftrightarrow \{V, W, \dots\}$ and $V \leftrightarrow U, W \leftrightarrow U, \dots$ by $\{V, W, \dots\} \leftrightarrow U$.

Example 2.3.3. $S \triangleright T \leftrightarrow T \triangleleft S$ is a translation rule \hat{l} from $\hat{\mathcal{G}}$ to $\hat{\mathcal{H}} = [\{s, t\}, \{a, b, c, \triangleleft\}, \{s \mapsto t \mid s \triangleleft s, t \mapsto a \mid b \mid c\}, s]$. Applications of \hat{l} to the string $a \triangleright b \triangleright c$ include both $a \triangleright c \triangleleft b$ and $b \triangleleft a \triangleright c$.

A *partial translation* of a statement $S \in \Lambda(\mathcal{G})$ under a set \mathbb{L} of translation rules between \mathcal{G} and \mathcal{H} is defined recursively as either S or an application of a translation rule $l \in \mathbb{L}$ to a partial translation of S . A statement $T \in \Lambda(\mathcal{H})$ that is a partial translation of S under \mathbb{L} is called an *translation* of S in \mathcal{H} under \mathbb{L} , in which case we write $S \rightsquigarrow_{\mathbb{L}} T$. Similarly, we write $\mathbb{V} \rightsquigarrow_{\mathbb{L}} \mathbb{W}$ to indicate that each statement $V \in \mathbb{V} \subseteq \Lambda(\mathcal{G})$ has at least one translation W in \mathcal{H} under \mathbb{L} and that $W \in \mathbb{W} \subseteq \Lambda(\mathcal{H})$ for any such translation W .

The set \mathbb{L} is called a *translator* from \mathcal{G} to \mathcal{H} when $\Lambda(\mathcal{G}) \rightsquigarrow_{\mathbb{L}} \Lambda(\mathcal{H})$. When such a set \mathbb{L} exists, we say that \mathcal{G} and \mathcal{H} are *equivalent under translation*. The set $\mathbb{L} = \{V_i \leftrightarrow W_i\}_{i \in (n)}$ for some $n \in \mathbb{N}$ is called a translator *between* grammars \mathcal{G} and \mathcal{H} when \mathbb{L} is a translator from \mathcal{G} to \mathcal{H} and $\mathbb{L}^{-1} \equiv \{W_i \leftrightarrow V_i\}_{i \in (n)}$ is a translator from \mathcal{H} to \mathcal{G} . Then we write $\mathbb{V} \leftrightarrow_{\mathbb{L}} \mathbb{W}$ to indicate that $\mathbb{V} \rightsquigarrow_{\mathbb{L}} \mathbb{W}$ and $\mathbb{W} \rightsquigarrow_{\mathbb{L}^{-1}} \mathbb{V}$.

When \mathbb{L} is a translator from \mathcal{G} to \mathcal{H} such that each statement $S \in \Lambda(\mathcal{G})$ has only one translation in \mathcal{H} under \mathbb{L} , we can define the *translation function* $\mathcal{L} : \Lambda(\mathcal{G}) \rightarrow \Lambda(\mathcal{H})$ of \mathbb{L} such that $\mathcal{L}(S)$ is the translation of S in \mathcal{H} under \mathbb{L} . A translator may be defined in terms of its translation function, given

recursively by declarations of the form $\mathcal{L}(V) = W$ for translation rules $V \rightsquigarrow W$ between \mathcal{G} and \mathcal{H} , where W is the translation of V in \mathcal{H} after any schematic variables in W have been substituted and any expressions $\mathcal{L}(\cdot)$ in W have been evaluated. In this case \mathcal{L} is called a translation function from \mathcal{G} to \mathcal{H} .

Example 2.3.4. The set $\hat{\mathbb{L}} = \{\hat{l}\}$ is a translator between $\hat{\mathcal{G}}$ and $\hat{\mathcal{H}}$. The translation function $\hat{\mathcal{L}}$ of $\hat{\mathbb{L}}$ may be defined by $\hat{\mathcal{L}}(S \triangleright T) = \hat{\mathcal{L}}(T) \triangleleft \hat{\mathcal{L}}(S)$.

Chapter 3

The MDPL Language

This chapter gives the formal specification of the MDPL language. The basic entity representing a computable function in MDPL is called a *model*, described in §3.1. MDPL models have an intuitive visual representation introduced in §3.2 that clarifies their functional interpretation. This interpretation is made explicit by rules for evaluating MDPL models given in §3.3. Finally, §3.4 shows that all computable functions can be expressed by MDPL models, that is, MDPL is universal.

3.1 Formal Definition

The basic entity in MDPL is a *model* \mathcal{M} over a countable set \mathbb{P} of *primitives*, defined recursively by $\mathcal{M} \equiv [\mathcal{Q}_{\mathcal{M}}, r_{\mathcal{M}}, \mathbb{S}_{\mathcal{M}}, \mu_{\mathcal{M}}, \nu_{\mathcal{M}}, \pi_{\mathcal{M}}, \rho_{\mathcal{M}}]$ consisting of

- a set $\mathcal{Q}_{\mathcal{M}}$ of models called *submodels*
- a symbol $r_{\mathcal{M}}$ called the *root node*
- a set $\mathbb{S}_{\mathcal{M}}$ of symbols called *subnodes*
- a function $\mu_{\mathcal{M}} : \mathbb{U}_{\mathcal{M}} \rightarrow \mathbb{Z}^*$ where $\mu_{\mathcal{M}}(s)$ is called the *arity* of $s \in \mathbb{S}_{\mathcal{M}}$ and $n_{\mathcal{M}} \equiv \mu_{\mathcal{M}}(r_{\mathcal{M}})$ is called the *coarity* of \mathcal{M}
- a function $\nu_{\mathcal{M}} : \mathbb{U}_{\mathcal{M}} \rightarrow \mathbb{Z}^*$ where $\nu_{\mathcal{M}}(s)$ is called the *coarity* of $s \in \mathbb{S}_{\mathcal{M}}$ and $m_{\mathcal{M}} \equiv \nu_{\mathcal{M}}(r_{\mathcal{M}})$ is called the *arity* of \mathcal{M}
- a function $\pi_{\mathcal{M}} : \mathbb{U}_{\mathcal{M}} \rightarrow \mathbb{P} \sqcup \mathcal{Q}_{\mathcal{M}} \sqcup \mathbb{U}_{\mathcal{M}} \sqcup \mathbb{X}_{\mathcal{M}} \sqcup \{\emptyset\}$ with $\pi_{\mathcal{M}}(r_{\mathcal{M}}) \in \mathbb{U}_{\mathcal{M}} \sqcup \mathbb{X}_{\mathcal{M}} \sqcup \{\emptyset\}$ and $\mathcal{Q}_{\mathcal{M}} \subseteq \pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}})$ where $\pi_{\mathcal{M}}(s)$ is called the *applicand* of the node $s \in \mathbb{U}_{\mathcal{M}}$
- a function $\rho_{\mathcal{M}} : \mathbb{Y}_{\mathcal{M}} \rightarrow \mathbb{P} \sqcup \mathbb{U}_{\mathcal{M}} \sqcup \mathbb{X}_{\mathcal{M}} \sqcup \{\emptyset\}$ where $\rho_{\mathcal{M}}(s, k)$ is called the *argument* of the sink $[s, k] \in \mathbb{Y}_{\mathcal{M}}$

where

- $\mathbb{U}_{\mathcal{M}} \equiv \mathbb{S}_{\mathcal{M}} \sqcup \{r_{\mathcal{M}}\}$ is called the set of *nodes*
- $\mathbb{X}_{\mathcal{M}} \equiv \bigsqcup_{s \in \mathbb{U}_{\mathcal{M}}} \{s\} \times \langle\langle \nu_{\mathcal{M}}(s) \rangle\rangle$ is called the set of *sources*
- $\mathbb{Y}_{\mathcal{M}} \equiv \bigsqcup_{s \in \mathbb{U}_{\mathcal{M}}} \{s\} \times \langle\langle -\mu_{\mathcal{M}}(s) \rangle\rangle$ is called the set of *sinks*
- $\chi_{\mathcal{M}} : \mathbb{X}_{\mathcal{M}} \longrightarrow \mathbb{U}_{\mathcal{M}}$ maps sources to their nodes, $\chi_{\mathcal{M}}(s, k) \equiv s$
- $\psi_{\mathcal{M}} : \mathbb{Y}_{\mathcal{M}} \longrightarrow \mathbb{U}_{\mathcal{M}}$ maps sinks to their nodes, $\psi_{\mathcal{M}}(s, k) \equiv s$
- $\omega_{\mathcal{M}} : \mathbb{X}_{\mathcal{M}} \sqcup \mathbb{Y}_{\mathcal{M}} \longrightarrow \langle\langle \nu_{\mathcal{M}}(s) \rangle\rangle \sqcup \langle\langle -\mu_{\mathcal{M}}(s) \rangle\rangle$ maps sources and sinks to their indices, $\omega_{\mathcal{M}}(s, k) \equiv k$
- $\mathfrak{H}_{\mathcal{M}}$ is a digraph called the *link digraph* with vertices $\mathbb{V}_{\mathcal{M}} \equiv \mathbb{P} \sqcup \mathbb{Q}_{\mathcal{M}} \sqcup \mathbb{U}_{\mathcal{M}} \sqcup \mathbb{X}_{\mathcal{M}} \sqcup \mathbb{Y}_{\mathcal{M}}$ and edges such that
 - primitives and submodels have no parents, $N_{\mathfrak{H}_{\mathcal{M}}}^-(\mathbb{P}) = N_{\mathfrak{H}_{\mathcal{M}}}^-(\mathbb{Q}_{\mathcal{M}}) = \emptyset$
 - nodes are children of their applicands and sinks, $\forall s \in \mathbb{U}_{\mathcal{M}} N_{\mathfrak{H}_{\mathcal{M}}}^-(s) = \{\pi_{\mathcal{M}}(s)\} \sqcup \{\mathcal{Y} \in \mathbb{Y}_{\mathcal{M}} \mid \psi_{\mathcal{M}}(\mathcal{Y}) = s\}$
 - sources are children of their nodes, $\forall \mathcal{X} \in \mathbb{X}_{\mathcal{M}} N_{\mathfrak{H}_{\mathcal{M}}}^-(\mathcal{X}) = \{\chi_{\mathcal{M}}(\mathcal{X})\}$
 - sinks are children of their arguments, $\forall \mathcal{Y} \in \mathbb{Y}_{\mathcal{M}} N_{\mathfrak{H}_{\mathcal{M}}}^-(\mathcal{Y}) = \{\rho_{\mathcal{M}}(\mathcal{Y})\}$

satisfying the following:

Causality Condition. *All cycles in $\mathfrak{H}_{\mathcal{M}}$ contain $r_{\mathcal{M}}$, that is, the restricted link graph $\underline{\mathfrak{H}}_{\mathcal{M}} \equiv \mathfrak{H}_{\mathcal{M}}[\mathbb{V}_{\mathcal{M}} \setminus \{r_{\mathcal{M}}\}]$ is a DAG.*

In §3.3 we define the interpretation of MDPL models as programs. There the above definitions can be interpreted intuitively as follows:

- each model \mathcal{M} defines a computable function
- the primitives \mathbb{P} represent distinguishable atomic entities (constants)
- the submodels $\mathbb{Q}_{\mathcal{M}}$ represent subfunctions defined within \mathcal{M}
- the root node $r_{\mathcal{M}}$ represents the function of \mathcal{M}
- the subnodes $\mathbb{S}_{\mathcal{M}}$ represents function calls within \mathcal{M}
- the arity $m_{\mathcal{M}}$ and coarity $n_{\mathcal{M}}$ represent the number of inputs and outputs of \mathcal{M} , respectively
- the functions $\mu_{\mathcal{M}}$ and $\nu_{\mathcal{M}}$ additionally define the number of inputs supplied to and outputs collected from the functions called at each subnode $s \in \mathbb{S}_{\mathcal{M}}$, respectively







- the sources $\mathbb{X}_{\mathcal{M}}$ refer to the inputs collected from $r_{\mathcal{M}}$ and the outputs collected from each $s \in \mathbb{S}_{\mathcal{M}}$
- the sinks $\mathbb{Y}_{\mathcal{M}}$ refer to the inputs supplied to each $s \in \mathbb{S}_{\mathcal{M}}$ and the outputs supplied to $r_{\mathcal{M}}$
- the function $\pi_{\mathcal{M}}$ defines the function called at each node $s \in \mathbb{U}_{\mathcal{M}}$
- the function $\rho_{\mathcal{M}}$ defines the value supplied to each sink, which may be undefined
- the digraph $\mathfrak{H}_{\mathcal{M}}$ indicates the dependencies in the evaluation of \mathcal{M} .



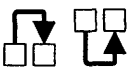
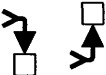



The causality condition may then be interpreted as stating that no part of the definition of \mathcal{M} is dependent upon itself in the evaluation of \mathcal{M} .

The formalism of MDPL is more complicated than that of many theoretical models of computation studied in computer science. However, it permits many kinds of operations on programs to be performed in a simple way, which would be much more complicated in simpler theoretical models. By building a programming language on MDPL, we avoid the need for many additional features required to compensate for the weaknesses of programming languages and environments built on simpler foundations. Indeed, many capabilities useful in the iterative development of programs arise naturally in MDPL (Chapter 5). At the same time, the difficulties of the MDPL formalism can be managed through a combination of visual representation (§3.2) and algorithmic transformations (Chapter 4) of MDPL models.






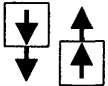
3.2 Visual Representation

We now introduce a visual representation of a model \mathcal{M} , in which:

- the root node $r_{\mathcal{M}}$ is represented by a rectangle 
- each subnode $s \in \mathbb{S}_{\mathcal{M}}$ is represented by a smaller rectangle inside the rectangle of $r_{\mathcal{M}}$ 
- the $m_{\mathcal{M}}$ sources of the root node, $[r_{\mathcal{M}}, 1], \dots, [r_{\mathcal{M}}, m_{\mathcal{M}}]$, are represented by a sequence of brackets from the left side of its rectangle from top to bottom 
- the $n_{\mathcal{M}}$ sinks of the root node, $[r_{\mathcal{M}}, -1], \dots, [r_{\mathcal{M}}, -n_{\mathcal{M}}]$, are represented by a sequence of brackets to the right side of its rectangle from top to bottom 
- the $\mu_{\mathcal{M}}(s)$ sinks of each $s \in \mathbb{S}_{\mathcal{M}}$, $[s, -1], \dots, [s, -\mu_{\mathcal{M}}(s)]$, are represented by a sequence of brackets to the left side of its rectangle from top to bottom 
- the $\nu_{\mathcal{M}}(s)$ sources of each $s \in \mathbb{S}_{\mathcal{M}}$, $[s, 1], \dots, [s, \nu_{\mathcal{M}}(s)]$, are represented by a sequence of brackets from the right side of its rectangle from top to bottom 

- a primitive $p \in \mathbb{P}$ is represented by a rectangle labeled with its symbol, with a single bracket on its right side 
- a submodel or primitive applicand $\mathcal{S} \in \mathbb{P} \sqcup \mathcal{Q}_{\mathcal{M}}$ of $s \in \mathbb{S}_{\mathcal{M}}$, $\pi_{\mathcal{M}}(s) = \mathcal{S}$, is indicated by superimposing the visual representation of \mathcal{S} on the rectangle of s when $\mathcal{S} \in \mathcal{Q}_{\mathcal{M}}$ (see below), or by labeling the rectangle of s with its symbol 
- a node applicand $a \in \mathbb{U}_{\mathcal{M}}$ of $s \in \mathbb{U}_{\mathcal{M}}$, $\pi_{\mathcal{M}}(s) = a$, is indicated by an arrow from the top or bottom of the rectangle of a to the top or bottom of the rectangle of s 
- a source applicand $\mathcal{X} \in \mathbb{X}_{\mathcal{M}}$ of $s \in \mathbb{S}_{\mathcal{M}}$, $\pi_{\mathcal{M}}(s) = \mathcal{X}$, is indicated by an arrow from the bracket of x to the top or bottom of the rectangle of s 
- a primitive argument $p \in \mathbb{P}$ of $\mathcal{Y} \in \mathbb{Y}_{\mathcal{M}}$, $\rho_{\mathcal{M}}(\mathcal{Y}) = p$, is indicated by an arrow from the bracket of the rectangle of p into the bracket of y 
- a node argument $a \in \mathbb{U}_{\mathcal{M}}$ of $\mathcal{Y} \in \mathbb{Y}_{\mathcal{M}}$, $\rho_{\mathcal{M}}(\mathcal{Y}) = a$, is indicated by an arrow from the top or bottom of the rectangle of a into the bracket of \mathcal{Y} 
- a source argument $\mathcal{X} \in \mathbb{X}_{\mathcal{M}}$ of $\mathcal{Y} \in \mathbb{Y}_{\mathcal{M}}$, $\rho_{\mathcal{M}}(\mathcal{Y}) = \mathcal{X}$, is indicated by an arrow from the bracket of x into the bracket of y . 

When the visual representation of a submodel $\mathcal{S} \in \mathcal{Q}_{\mathcal{M}}$ is superimposed on the rectangle of a subnode $s \in \mathbb{S}_{\mathcal{M}}$:

- the rectangle of $r_{\mathcal{S}}$ coincides with the rectangle of s 
- the first \bar{m} brackets on their left sides are aligned from top to bottom, where $\bar{m} = \min\{\mu_{\mathcal{M}}(s), m_{\mathcal{S}}\}$, with the remaining brackets appearing below 
- the first \bar{n} brackets on their right sides are aligned from top to bottom, where $\bar{n} = \min\{\nu_{\mathcal{M}}(s), n_{\mathcal{S}}\}$, with the remaining brackets appearing below 
- when a sink $[s, -i] \in \mathbb{Y}_{\mathcal{M}}$ of \mathcal{M} for $i \in \langle \bar{m} \rangle$ is defined, $\rho_{\mathcal{M}}(s, -i) \neq \emptyset$, the bracket of $[s, -i]$ is omitted, and the arrow into it is drawn into the bracket of the source $[r_{\mathcal{S}}, i] \in \mathbb{X}_{\mathcal{S}}$ of \mathcal{S} 
- when a sink $[r_{\mathcal{S}}, -j] \in \mathbb{Y}_{\mathcal{S}}$ of \mathcal{S} for $j \in \langle \bar{n} \rangle$ is defined, $\rho_{\mathcal{S}}(r_{\mathcal{S}}, -j) \neq \emptyset$, the bracket of $[r_{\mathcal{S}}, -j]$ is omitted, and the arrow into it is drawn into the bracket of the source $[s, j] \in \mathbb{X}_{\mathcal{M}}$ of \mathcal{M} 
- when an applicand of the root node of \mathcal{S} is defined, $\pi_{\mathcal{S}}(r_{\mathcal{S}}) \neq \emptyset$, the head of the arrow to the rectangle of $r_{\mathcal{S}}$ is aligned with the tail of any arrow from the rectangle of s in \mathcal{M} 

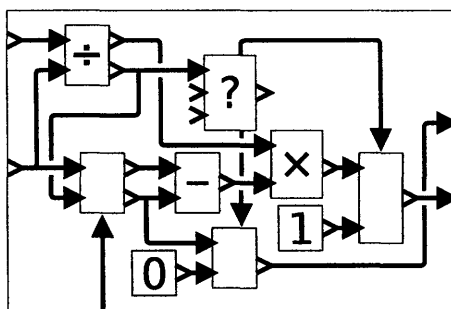


Figure 3-1: An MDPL model representing the extended GCD function

Example 3.2.1. The MDPL model in Figure 3-1 represents the extended GCD function: given two nonzero integers $a, b \in \mathbb{Z}^\circ$, it returns two integers $x, y \in \mathbb{Z}$ such that $a \cdot x + b \cdot y = \text{GCD}(a, b)$, where $\text{GCD}(a, b)$ is the greatest common divisor of a and b . In this interpretation, the submodels in the definition represent:

- the integer division function: given an integer $u \in \mathbb{Z}$ and a nonzero integer $v \in \mathbb{Z}^\circ$, it returns two integers $q, r \in \mathbb{Z}$ such that $q = \lfloor \frac{u}{v} \rfloor$ is the integer quotient of u and v and $r = u - q \cdot v$ is the integer remainder of u and v
- the integer multiplication function: given two integers $u, v \in \mathbb{Z}$, it returns their product $u \cdot v \in \mathbb{Z}$
- the integer subtraction function: given two integers $u, v \in \mathbb{Z}$, it returns their difference $u - v \in \mathbb{Z}$
- the zero condition function: given three inputs x, y, z , it returns y when $x \neq 0$ and z when $x = 0$



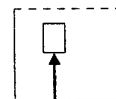
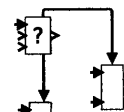
and the primitives in the definition represent:

- the integer constant 0
- the integer constant 1.



The definition of the model is based on the extended Euclidean algorithm [73]. Note the following interpretations made in evaluating the model (see §3.3):

- currying: the zero condition function of three arguments with one defined argument specifies a function of two arguments, which is applied to two of the subnodes in the model, where the remaining two arguments are defined independently in each case
- recursion: the root node is the applicand of one of the subnodes in the model, indicating that the function of the model should be applied recursively at that subnode



Explicit recursion is not strictly necessary to implement recursive functions (see §3.4.3), but this example demonstrates how it can facilitate expression in MDPL, as in most modern programming languages.

3.3 Model Evaluation

The interpretation of MDPL models as computable functions is defined by a set of transformation rules in §3.3.1 that can be applied to a model to yield another model. We will write $\mathcal{M} \rightarrow \mathcal{N}$ to indicate that the model \mathcal{N} is obtained from the model \mathcal{M} by applying one of the transformations. We write $\mathcal{M} \twoheadrightarrow \mathcal{N}$ to indicate that \mathcal{N} is obtained from \mathcal{M} by repeatedly applying zero or more transformations. That is, \twoheadrightarrow is the reflexive, transitive closure of \rightarrow .

The argument outlined in §3.3.2 shows that the relation \twoheadrightarrow is *confluent*, that is, if $\mathcal{M} \twoheadrightarrow \mathcal{N}$ and $\mathcal{M} \twoheadrightarrow \mathcal{N}'$, then there exists a model \mathcal{R} such that $\mathcal{N} \twoheadrightarrow \mathcal{R}$ and $\mathcal{N}' \twoheadrightarrow \mathcal{R}$. It follows that there is an equivalence relation $\twoheadrightarrow\!\!\!\leftarrow$ on models called *normalization*, defined by $\mathcal{N} \twoheadrightarrow\!\!\!\leftarrow \mathcal{N}'$ when there exists \mathcal{R} such that $\mathcal{N} \twoheadrightarrow \mathcal{R}$ and $\mathcal{N}' \twoheadrightarrow \mathcal{R}$. For if $\mathcal{M} \twoheadrightarrow\!\!\!\leftarrow \mathcal{M}' \twoheadrightarrow\!\!\!\leftarrow \mathcal{M}''$, there exist $\mathcal{N}, \mathcal{N}'$ such that $\mathcal{M}, \mathcal{M}' \twoheadrightarrow \mathcal{N}$ and $\mathcal{M}', \mathcal{M}'' \twoheadrightarrow \mathcal{N}'$, so the preceding implies there exists \mathcal{R} such that $\mathcal{N}, \mathcal{N}' \twoheadrightarrow \mathcal{R}$, which implies $\mathcal{M} \twoheadrightarrow\!\!\!\leftarrow \mathcal{M}''$.

A *normal form* is a model \mathcal{R} to which none of the transformation rules can be applied, that is, where $\mathcal{N} \twoheadrightarrow \mathcal{R}$ if and only if $\mathcal{N} = \mathcal{R}$. For two normal forms $\mathcal{R}, \mathcal{R}'$, this means $\mathcal{R} \twoheadrightarrow\!\!\!\leftarrow \mathcal{R}'$ if and only if $\mathcal{R} = \mathcal{R}'$, that is, each equivalence class under normalization contains at most one unique normal form, so every model can be normalized to at most one normal form. As shown in §3.3.3, any model equivalent to a normal form can be transformed to that normal form by a terminating algorithm repeatedly applying the transformation rules: we call this process *normal form reduction*. This defines a consistent functional interpretation of MDPL models.

3.3.1 The Normalization Transformations

An *internal model* \mathcal{M} of a model \mathcal{N} is defined recursively as either \mathcal{N} or an internal model of some submodel $S \in \mathcal{Q}_{\mathcal{N}}$. We indicate the set of all internal models of \mathcal{M} by $\mathcal{R}_{\mathcal{M}}$. Under each of the following headers, we define a model \mathcal{M}' based on a given model \mathcal{M} satisfying certain conditions. For each case, when $\mathcal{M} \in \mathcal{R}_{\mathcal{N}}$ is an internal model of \mathcal{N} , we declare $\mathcal{N} \rightarrow \mathcal{N}'$ where \mathcal{N}' is the model obtained from \mathcal{N} by replacing a particular instance of \mathcal{M} by \mathcal{M}' . To indicate a specific transformation XYZ, we write $\mathcal{N} \xrightarrow{\text{XYZ}} \mathcal{N}'$. These transformations completely define the normalization relation for MDPL models.

For convenience, we introduce an extension of $\rho_{\mathcal{M}}$ defined by $\bar{\rho}_{\mathcal{M}}(s, k) \equiv \begin{cases} \rho_{\mathcal{M}}(s, k) & k \in \langle -\mu_{\mathcal{M}}(s) \rangle \\ \emptyset & \text{otherwise} \end{cases}$ for $s \in \mathbb{U}_{\mathcal{M}}, k \in \mathbb{Z}^-$, that is explicitly undefined for all negative indices k for which there are no corresponding sinks. We can then define $\varrho_{\mathcal{M}}(s, k) \equiv |\{l \in \langle k \rangle \mid \bar{\rho}_{\mathcal{M}}(s, l) = \emptyset\}|$, which counts the number of indices l , up to the given index k , for which $\bar{\rho}_{\mathcal{M}}(s, l)$ is undefined. We abbreviate $\varrho_{\mathcal{M}}(s) \equiv \varrho_{\mathcal{M}}(s, -\mu_{\mathcal{M}}(s))$ to indicate the total number of sinks of $s \in \mathbb{U}_{\mathcal{M}}$ with undefined arguments.

Subnode Elimination (SNL) The model \mathcal{M} has a subnode $a \in \mathbb{S}_{\mathcal{M}}$ such that no source of a nor a itself is an applicand or argument of any node or sink in \mathcal{M} , that is, $a \notin \pi_{\mathcal{M}}(\mathbb{U}_{\mathcal{M}}) \cup \rho_{\mathcal{M}}(\mathbb{Y}_{\mathcal{M}}) \cup \chi_{\mathcal{M}}(\rho_{\mathcal{M}}(\mathbb{Y}_{\mathcal{M}}) \cap \mathbb{X}_{\mathcal{M}})$.

Then the model \mathcal{M}' in $\mathcal{N}' = \tau_a^{\text{SNL}}(\mathcal{M})$ is constructed by removing the subnode a from \mathcal{M} ,

$$\begin{aligned}\mathcal{Q}_{\mathcal{M}'} &= \pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}} \setminus \{a\}) \cap \mathcal{Q}_{\mathcal{M}} \\ r_{\mathcal{M}'} &= r_{\mathcal{M}} \\ \mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \setminus \{a\} \\ \mu_{\mathcal{M}'}(s) &= \mu_{\mathcal{M}}(s) \\ \nu_{\mathcal{M}'}(s) &= \nu_{\mathcal{M}}(s) \\ \pi_{\mathcal{M}'}(s) &= \pi_{\mathcal{M}}(s) \\ \rho_{\mathcal{M}'}(\mathcal{Y}) &= \rho_{\mathcal{M}}(\mathcal{Y}),\end{aligned}$$

leaving the remaining nodes unchanged. This transformation effectively removes the subnode a when it has no effect on the functional interpretation of \mathcal{M} . Hence we say that it *eliminates* a . In the visual representation, the rectangle of a is removed along with its contents and any adjoining (incoming) arrows to it or sinks on its outside.

Subnode Expansion (SNX) The model \mathcal{M} has a subnode $a \in \mathbb{S}_{\mathcal{M}}$ for which there is either

1. a subnode $t \in \mathbb{S}_{\mathcal{M}}$ with applicand $\pi_{\mathcal{M}}(t) = a$, or
2. a sink $\mathcal{T} \in \mathbb{Y}_{\mathcal{M}}$ with argument $\rho_{\mathcal{M}}(\mathcal{T}) = a$.

Then the model \mathcal{M}' in $\mathcal{N}' = \tau_t^{\text{SNX}}(\mathcal{M})$ or $\mathcal{N}' = \tau_{\mathcal{T}}^{\text{SNX}}(\mathcal{M})$ is constructed by adding an additional subnode a' to \mathcal{M} ,

$$\begin{aligned}\mathcal{Q}_{\mathcal{M}'} &= \mathcal{Q}_{\mathcal{M}} \\ r_{\mathcal{M}'} &= r_{\mathcal{M}} \\ \mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \sqcup \{a'\},\end{aligned}$$

where either

1. the subnode a' has the same arity, coarity, applicand, and arguments as a , and the subnode t

has applicand a' ,

$$\begin{aligned}\mu_{\mathcal{M}'}(s) &= \begin{cases} \mu_{\mathcal{M}}(a) & s = a' \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \nu_{\mathcal{M}'}(s) &= \begin{cases} \nu_{\mathcal{M}}(a) & s = a' \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \pi_{\mathcal{M}'}(s) &= \begin{cases} a' & s = t \\ \pi_{\mathcal{M}}(a) & s = a' \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \rho_{\mathcal{M}'}(s, k) &= \begin{cases} \rho_{\mathcal{M}}(a, k) & s = a' \\ \rho_{\mathcal{M}}(s, k) & \text{otherwise} \end{cases},\end{aligned}$$

or

2. the subnode a' has zero arity/coarity and applicand a , and the sink \mathcal{T} has argument a' ,

$$\begin{aligned}\mu_{\mathcal{M}'}(s) &= \begin{cases} 0 & s = a' \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \nu_{\mathcal{M}'}(s) &= \begin{cases} 0 & s = a' \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \pi_{\mathcal{M}'}(s) &= \begin{cases} a & s = a' \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \rho_{\mathcal{M}'}(\mathcal{Y}) &= \begin{cases} a' & \mathcal{Y} = \mathcal{T} \\ \rho_{\mathcal{M}}(\mathcal{Y}) & \text{otherwise} \end{cases},\end{aligned}$$

respectively. This transformation effectively duplicates the subnode a as a' , where a' takes the place of a as the applicand or argument to some subnode or sink, respectively. Hence we say it *expands a with respect to t or \mathcal{T} (respectively) as a'* . In the visual representation, either the rectangle of a is duplicated along with its contents and any incoming arrows to it or sinks on its outside to form the rectangle of a' , or the outgoing arrow from the rectangle of a is redirected to the new empty rectangle of a' ; and the tail of the outgoing arrow from the rectangle of a to the rectangle of t or one of the sinks on its outside is transferred from the rectangle of a to the rectangle of a' .

Subnode Application (SNA) The model \mathcal{M} has a subnode $t \in \mathbb{S}_{\mathcal{M}}$ with subnode applicand $\pi_{\mathcal{M}}(t) = a \in \mathbb{S}_{\mathcal{M}}$, and either

1. the applicand of a is a submodel $\pi_{\mathcal{M}}(a) = \mathcal{S} \in \mathcal{Q}_{\mathcal{M}}$ whose root node has a defined applicand $\pi_{\mathcal{S}}(r_{\mathcal{S}}) \neq \emptyset$ or
2. the applicand of a is not a node or source, $\pi_{\mathcal{M}}(a) \notin \mathbb{U}_{\mathcal{M}} \sqcup \mathbb{X}_{\mathcal{M}}$.

Then the model \mathcal{M}' in $\mathcal{N}' = \tau_t^{\text{SNA}}(\mathcal{N})$ is constructed by reassigning the arguments of sinks of t in \mathcal{M}' to be the arguments obtained by “filling” the sinks of a with the arguments of sinks of t : beginning with the arguments of sinks of a , successively replacing those undefined with arguments of sinks of t , and then successively appending the remaining arguments of sinks of t as arguments of additional sinks of a ,

$$\begin{aligned} \mathcal{Q}_{\mathcal{M}'} &= \mathcal{Q}_{\mathcal{M}} \\ r_{\mathcal{M}'} &= r_{\mathcal{M}} \\ \mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \\ \mu_{\mathcal{M}'}(s) &= \begin{cases} \mu_{\mathcal{M}}(a) + \max\{0, \mu_{\mathcal{M}}(t) - \varrho_{\mathcal{M}}(a)\} & s = t \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases}, \end{aligned}$$

and by reassigning the applicand of t in \mathcal{M}' to be the applicand of a , and either

1. reassigning arguments and applicands equal to sources of t to the corresponding sources of t beyond those corresponding to sinks of $r_{\mathcal{S}}$,

$$\begin{aligned} \nu_{\mathcal{M}'}(s) &= \begin{cases} \nu_{\mathcal{M}}(t) + n_{\mathcal{S}} & s = t \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \pi_{\mathcal{M}'}(s) &= \begin{cases} [t, l + n_{\mathcal{S}}] & \pi_{\mathcal{M}}(s) = [t, l] \in \mathbb{X}_{\mathcal{M}} \\ \pi_{\mathcal{M}}(a) & s = t \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \rho_{\mathcal{M}'}(s, k) &= \begin{cases} [t, l + n_{\mathcal{S}}] & \rho_{\mathcal{M}}(s, k) = [t, l] \in \mathbb{X}_{\mathcal{M}} \\ \begin{cases} \bar{\rho}_{\mathcal{M}}(t, -\varrho_{\mathcal{M}}(a, k)) & \bar{\rho}_{\mathcal{M}}(a, k) = \emptyset \\ \rho_{\mathcal{M}}(a, k) & \text{otherwise} \end{cases} & s = t \\ \rho_{\mathcal{M}}(s, k) & \text{otherwise} \end{cases}, \end{aligned}$$

or

2. making no additional changes,

$$\begin{aligned} \nu_{\mathcal{M}'}(s) &= \nu_{\mathcal{M}}(s) \\ \pi_{\mathcal{M}'}(s) &= \begin{cases} \pi_{\mathcal{M}}(a) & s = t \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \rho_{\mathcal{M}'}(s, k) &= \begin{cases} \begin{cases} \bar{\rho}_{\mathcal{M}}(t, -\varrho_{\mathcal{M}}(a, k)) & \bar{\rho}_{\mathcal{M}}(a, k) = \emptyset \\ \rho_{\mathcal{M}}(a, k) & \text{otherwise} \end{cases} & s = t \\ \rho_{\mathcal{M}}(s, k) & \text{otherwise} \end{cases} \end{aligned}$$

This transformation effectively assigns the subfunction called at t to be the same as that of its applicand a with the substitution of arguments from a , consistent with the functional interpretation of \mathcal{M} . Hence we say it *applies* a to t . In the visual representation, the rectangle of t and its contents are replaced by a duplicate of the rectangle of a with its contents and any incoming arrows to it or sinks on its outside, and the incoming arrows to t are adjoined to the unjoined sinks on the rectangle of a , with excess arrows adjoined to additional sinks; and when the applicand of a is a submodel whose root node has an applicand other than itself, the arrows to the sources on the outside of the rectangle of t are shifted downwards to sources below the new corresponding sinks on the inside of the rectangle of t .

Root Node Expansion (RNX) The model \mathcal{M} has either

1. a node $t \in \mathbb{U}_{\mathcal{M}}$ with applicand $\pi_{\mathcal{M}}(t) = r_{\mathcal{M}}$
2. a sink $\mathcal{T} \in \mathbb{Y}_{\mathcal{M}}$ with argument $\rho_{\mathcal{M}}(\mathcal{T}) = r_{\mathcal{M}}$.

Then the model \mathcal{M}' in $\mathcal{N}' = \tau_t^{\text{RNX}}(\mathcal{N})$ or $\mathcal{N}' = \tau_{\mathcal{T}}^{\text{RNX}}(\mathcal{N})$ is constructed by either

1. adding the model \mathcal{M} as a submodel applicand of a new subnode applicand a of t ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= \mathcal{Q}_{\mathcal{M}} \cup \{\mathcal{M}\} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \sqcup \{a\} \\
\mu_{\mathcal{M}'}(s) &= \begin{cases} 0 & s = a \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}'}(s) &= \begin{cases} 0 & s = a \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}'}(s) &= \begin{cases} \mathcal{M} & s = a \\ a & s = t \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}'}(\mathcal{Y}) &= \rho_{\mathcal{M}}(\mathcal{Y}),
\end{aligned}$$

or

2. adding a subnode a to \mathcal{M} with applicand $r_{\mathcal{M}}$ and zero arity/coarity as the argument of \mathcal{T} ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= \mathcal{Q}_{\mathcal{M}} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \sqcup \{a\} \\
\mu_{\mathcal{M}'}(s) &= \begin{cases} 0 & s = a \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}'}(s) &= \begin{cases} 0 & s = a \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}'}(s) &= \begin{cases} r_{\mathcal{M}} & s = a \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}'}(\mathcal{Y}) &= \begin{cases} a & \mathcal{Y} = \mathcal{T} \\ \rho_{\mathcal{M}}(\mathcal{Y}) & \text{otherwise} \end{cases},
\end{aligned}$$

respectively. This transformation effectively either duplicates the functional interpretation of $r_{\mathcal{M}}$ within the subnode t to which the root node is applicand, or creates such a node as the argument of the sink \mathcal{T} with argument $r_{\mathcal{M}}$, respectively. Hence we say it *expands* $r_{\mathcal{M}}$ *with respect to* t or \mathcal{T} ,

respectively. In the visual representation, either the contents of the rectangle of \mathcal{M} are duplicated inside the rectangle of t , or there is drawn an arrow from the rectangle of the root node $r_{\mathcal{M}}$ to the rectangle of a new subnode a , and the tail of the arrow from the rectangle of $r_{\mathcal{M}}$ to the sink \mathcal{T} is transferred to the rectangle of a .

Submodel Application (SMA) The model \mathcal{M} has a subnode $a \in \mathbb{S}_{\mathcal{M}}$ with a submodel applicand $\pi_{\mathcal{M}}(a) = \mathcal{S} \in \mathcal{Q}_{\mathcal{M}}$ in which the root node $r_{\mathcal{S}}$ is neither the applicand of any node nor the argument of any sink in \mathcal{S} , $r_{\mathcal{S}} \notin \pi_{\mathcal{S}}(\mathbb{U}_{\mathcal{S}}) \cup \rho_{\mathcal{S}}(\mathbb{Y}_{\mathcal{S}})$. We distinguish the cases where

1. the applicand $\pi_{\mathcal{M}}(r_{\mathcal{M}}) = a$ and a is neither the applicand of any subnode nor the argument of any sink in \mathcal{M} , $a \notin \pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}}) \cup \rho_{\mathcal{M}}(\mathbb{Y}_{\mathcal{M}})$, or
2. the applicand $\pi_{\mathcal{M}}(r_{\mathcal{M}}) \neq a$, and either a is neither the applicand of any subnode nor the argument of any sink in \mathcal{M} , $a \notin \pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}}) \cup \rho_{\mathcal{M}}(\mathbb{Y}_{\mathcal{M}})$, or the arity $m_{\mathcal{S}} = 0$,

where in each case either

- (a) the applicand of $r_{\mathcal{S}}$ is undefined, $\pi_{\mathcal{S}}(r_{\mathcal{S}}) = \emptyset$, or
- (b) the applicand of $r_{\mathcal{S}}$ is a subnode or source of \mathcal{S} , $\pi_{\mathcal{S}}(r_{\mathcal{S}}) \in \mathbb{S}_{\mathcal{S}} \sqcup \mathbb{X}_{\mathcal{S}}$.

Then the model \mathcal{M}' in $\mathcal{N}' = \tau_a^{\text{SMA}}(\mathcal{N})$ is constructed by combining the subnodes of \mathcal{S} with the subnodes of \mathcal{M} ,

$$\begin{aligned} r_{\mathcal{M}'} &= r_{\mathcal{M}} \\ \mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \sqcup \mathbb{S}_{\mathcal{S}}, \end{aligned}$$

where subnodes are relabeled as needed so that $\mathbb{S}_{\mathcal{M}} \sqcup \mathbb{S}_{\mathcal{S}}$, and either

1. filling any undefined sinks of a with new sources of $r_{\mathcal{M}}$ and
 - (a) reassigning the applicands and arguments of \mathcal{S} that are sources of $r_{\mathcal{S}}$ to be the arguments of the corresponding sinks of a , and reassigning the applicands and arguments of \mathcal{M} that

are sources of a to be the arguments of the corresponding sinks of r_S ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= \mathcal{Q}_{\mathcal{M}} \\
\mu_{\mathcal{M}'}(s) &= \begin{cases} \mu_S(s) & s \in \mathbb{S}_S \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}'}(s) &= \begin{cases} m_{\mathcal{M}} + \varrho_{\mathcal{M}}(a, m_S) & s = r_{\mathcal{M}} \\ \nu_S(s) & s \in \mathbb{S}_S \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}'}(s) &= \begin{cases} \beta_{\mathcal{M},S}(a, \pi_S(s)) & s \in \mathbb{S}_S \\ \gamma_{\mathcal{M},S}(a, \pi_{\mathcal{M}}(s)) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}'}(s, k) &= \begin{cases} \alpha_{\mathcal{M}}(a, k) & s = a \\ \beta_{\mathcal{M},S}(a, \rho_S(s, k)) & s \in \mathbb{S}_S \\ \gamma_{\mathcal{M},S}(a, \rho_{\mathcal{M}}(s, k)) & \text{otherwise} \end{cases},
\end{aligned}$$

where

$$\begin{aligned}
\alpha_{\mathcal{M}}(s, k) &\equiv \begin{cases} [r_{\mathcal{M}}, m_{\mathcal{M}} + \varrho_{\mathcal{M}}(s, -k)] & \bar{\rho}_{\mathcal{M}}(s, k) = \emptyset \\ \bar{\rho}_{\mathcal{M}}(s, k) & \text{otherwise} \end{cases} \\
\beta_{\mathcal{M},S}(s, u) &\equiv \begin{cases} \alpha_{\mathcal{M}}(s, -l) & u = [r_S, l] \in \mathbb{X}_S \\ u & \text{otherwise} \end{cases} \\
\gamma_{\mathcal{M},S}(s, u) &\equiv \begin{cases} \beta_{\mathcal{M},S}(s, \bar{\rho}_S(r_S, -h)) & u = [s, h] \in \mathbb{X}_{\mathcal{M}} \\ u & \text{otherwise} \end{cases},
\end{aligned}$$

or

- (b) in addition to the above, reassigning arguments and applicands so that the sources and sinks of a in \mathcal{M}' correspond to those in \mathcal{M} without corresponding sinks and sources of

r_S , and reassigning the applicand of a to be the applicand of r_S ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= (\pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}} \setminus \{a\}) \cap \mathcal{Q}_{\mathcal{M}}) \cup \mathcal{Q}_S \\
\mu_{\mathcal{M}'}(s) &= \begin{cases} \max\{0, \mu_{\mathcal{M}}(a) - m_S\} & s = a \\ \mu_S(s) & s \in \mathbb{S}_S \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}'}(s) &= \begin{cases} \max\{0, \nu_{\mathcal{M}}(a) - n_S\} & s = a \\ m_{\mathcal{M}} + \varrho_{\mathcal{M}}(a, m_S) & s = r_{\mathcal{M}} \\ \nu_S(s) & s \in \mathbb{S}_S \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}'}(s) &= \begin{cases} \beta_{\mathcal{M},S}(a, \pi_S(r_S)) & s = a \\ \beta_{\mathcal{M},S}(a, \pi_S(s)) & s \in \mathbb{S}_S \\ \tilde{\gamma}_{\mathcal{M},S}(a, \pi_{\mathcal{M}}(s)) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}'}(s, k) &= \begin{cases} \rho_{\mathcal{M}}(a, k - m_S) & s = a \\ \beta_{\mathcal{M},S}(a, \rho_S(s, k)) & s \in \mathbb{S}_S \\ \tilde{\gamma}_{\mathcal{M},S}(a, \rho_{\mathcal{M}}(s, k)) & \text{otherwise} \end{cases},
\end{aligned}$$

where

$$\tilde{\gamma}_{\mathcal{M},S}(s, u) \equiv \begin{cases} \begin{cases} \beta_{\mathcal{M},S}(s, \bar{\rho}_S(r_S, -h)) & h \in \langle n_S \rangle \\ [s, h - n_S] & \text{otherwise} \end{cases} & u = [s, h] \in \mathbb{X}_{\mathcal{M}} \\ u & \text{otherwise} \end{cases},$$

or

2. defining \mathcal{M}' as above, but without filling undefined sinks of a with new sources of $r_{\mathcal{M}}$: that is, replacing α by $\tilde{\alpha}$ in the equations above where

$$\tilde{\alpha}_{\mathcal{M}}(s, k) \equiv \bar{\rho}_{\mathcal{M}}(s, k),$$

respectively. This transformation effectively embeds the submodel \mathcal{S} into the definition of \mathcal{M} , and modifies the subnode a to yield a functional interpretation in \mathcal{M} consistent with the applicands of r_S in \mathcal{S} and $r_{\mathcal{M}}$ in \mathcal{M} , carrying over the sinks and sources of a in \mathcal{M} not corresponding to sources and sinks of \mathcal{S} , respectively. Hence we say it *applies* the submodel \mathcal{S} at a . In the visual representation, the original rectangle of a (but not its contents) is removed and the arrows adjoined to sinks and

sources on its outside are joined with the arrows adjoined to the corresponding sources and sinks on its inside, and the rectangle of a is duplicated with either its contents and any incoming or outgoing arrows to it or sinks and sources on its outside, or the original incoming arrow to it and any incoming or outgoing arrows to sinks and sources on its outside that have no corresponding sources and sinks on its inside, to form the new rectangle of a . In typical cases, together with the other transformation rules above, this rule is equivalent to simply removing the rectangle of a and joining all aligned interior and exterior arrows adjoined to it or to the sources and sinks on its boundary.

The above set of normalization transformations is chosen to provide a straightforward description of the functional interpretation of MDPL models. It is not the only possible set of normalization transformations that can describe this interpretation, nor is it maximally restrictive in the transformation operations it permits. Note that in typical usage, the transformation equations reduce to quite simple expressions, since the full power of the language is scarcely required to define most ordinary functions.

For practical implementation, a programming language based on MDPL would typically augment the normalization transformations above with special normalization rules for certain primitives \mathbb{P} . Such rules would effectively define the behavior of “primitive functions” that can be most efficiently implemented directly (for instance, in hardware) external to the language. For example, practical programming languages based on the lambda calculus typically implement numbers and arithmetic operations directly using hardware instructions in whatever microprocessor is the target of compilation, rather than employing the relatively inefficient encoding of Church numerals and associated arithmetic operations in lambda calculus. Our prototype implementation of MDPL introduced in §5.1 uses some predefined primitives implemented directly in the prototype platform.

3.3.2 Confluence of the Normalization Transformations

Here we outline an argument establishing the confluence of the \rightarrow relation, that is: if $\mathcal{M} \rightarrow \mathcal{N}$ and $\mathcal{M} \rightarrow \mathcal{Q}$, then there exists a model \mathcal{R} such that $\mathcal{N} \rightarrow \mathcal{R}$ and $\mathcal{Q} \rightarrow \mathcal{R}$. This result is analogous to the Church-Rosser Theorem for λ calculus, though the argument is not, since the normalization transformations are entirely different. To simplify matters, we will omit the RNX transformation from the discussion, since it is not necessary for the core formalism of MDPL (see §3.4.3).

Let $C(n)$ be the proposition that such an \mathcal{R} exists whenever $\mathcal{M} \xrightarrow{n} \mathcal{N}$ and $\mathcal{M} \rightarrow \mathcal{Q}$, where $\mathcal{M} \xrightarrow{n} \mathcal{N}$ denotes $\mathcal{M} \underbrace{\rightarrow \cdots \rightarrow}_{n \text{ times}} \mathcal{N}$ for $n \in \mathbb{N}$, and suppose $C(1)$ holds. Now suppose $C(q)$ holds for some $q \in \mathbb{N}$. Let \mathcal{N} be any model such that $\mathcal{M} \xrightarrow{q+1} \mathcal{N}$, hence there is a model \mathcal{M}' such that $\mathcal{M} \xrightarrow{q} \mathcal{M}'$ and $\mathcal{M}' \rightarrow \mathcal{N}$. From $C(q)$, we know that when $\mathcal{M} \xrightarrow{q} \mathcal{M}'$ and $\mathcal{M} \rightarrow \mathcal{Q}$, there exists \mathcal{R}' for which $\mathcal{M}' \rightarrow \mathcal{R}'$ and $\mathcal{Q} \rightarrow \mathcal{R}'$. From $C(1)$, we know that when $\mathcal{M}' \rightarrow \mathcal{N}$ and $\mathcal{M}' \rightarrow \mathcal{R}'$, there exists \mathcal{R}

such that $\mathcal{N} \rightarrow \mathcal{R}$ and $\mathcal{R}' \rightarrow \mathcal{R}$. Thus given $\mathcal{M} \xrightarrow{q+1} \mathcal{N}$ and $\mathcal{M} \rightarrow \mathcal{Q}$, we have $\mathcal{N} \rightarrow \mathcal{R}$ and $\mathcal{Q} \rightarrow \mathcal{R}$, hence $C(q+1)$. By induction on the number of transformations q , $C(1)$ implies $C(n)$ for all $n \in \mathbb{N}$, which proves the confluence of the \rightarrow relation. Hence we need only verify $C(1)$: if $\mathcal{M} \rightarrow \mathcal{N}$ and $\mathcal{M} \rightarrow \mathcal{Q}$, then there exists a model \mathcal{R} such that $\mathcal{N} \rightarrow \mathcal{R}$ and $\mathcal{Q} \rightarrow \mathcal{R}$.

To prove $C(1)$, we consider each transformation XYZ among the normalization transformations, and suppose $\mathcal{M} \xrightarrow{XYZ} \mathcal{N}$ for an arbitrary model \mathcal{M} , in each possible way. In each of these cases, we consider each possible transformation $\mathcal{M} \rightarrow \mathcal{M}'$, and show that there are corresponding transformations $\mathcal{N} \rightarrow \mathcal{N}'$ (independent of the original XYZ transformation) such that $\mathcal{M}' \rightarrow \mathcal{N}'$. By a similar induction on the number of transformations, we establish the same holds for an arbitrarily long sequence of transformations $\mathcal{M} \rightarrow \mathcal{M}'$, which establishes $C(1)$.

For a subset $\mathbb{T} \subseteq \mathbb{S}_{\mathcal{M}}$ of the subnodes of some model \mathcal{M} , consider a sequence of SNX transformations $\mathcal{M} \xrightarrow{\text{SNX}} \dots \xrightarrow{\text{SNX}} \mathcal{M}'$ where each transformation expands some subnode $s \in \mathbb{T}$, no transformation creates a subnode that can be eliminated by the SNL transformation, and no SNX transformation can expand a subnode $s \in \mathbb{T}$ in \mathcal{M}' without creating such a subnode. We will say that this sequence of transformations *fully expands* the subnodes \mathbb{T} .

For a given transformation $\mathcal{M} \xrightarrow{XYZ} \mathcal{N}$, the outline of the argument is as follows:

1. if the transformation $\mathcal{M} \rightarrow \mathcal{M}'$ is identical to the transformation $\mathcal{M} \xrightarrow{XYZ} \mathcal{N}$, then $\mathcal{N} = \mathcal{M}'$ and we are done. This is the trivial case; otherwise:
2. if the transformation $\mathcal{M} \rightarrow \mathcal{M}'$ maintains all values of $\mu_{\mathcal{M}}, \nu_{\mathcal{M}}, \pi_{\mathcal{M}}, \rho_{\mathcal{M}}$ changed by $\mathcal{M} \xrightarrow{XYZ} \mathcal{N}$, transform $\mathcal{N} \rightarrow \mathcal{N}'$ by the same transformation as $\mathcal{M} \rightarrow \mathcal{M}'$, then $\mathcal{M}' \xrightarrow{XYZ} \mathcal{N}'$ by the same transformation as $\mathcal{M} \xrightarrow{XYZ} \mathcal{N}$. This includes the cases where the transformations $\mathcal{M} \xrightarrow{XYZ} \mathcal{N}$ and $\mathcal{M} \rightarrow \mathcal{M}'$ occur in different submodels, or occur in the same submodel but do not interact; otherwise:
3. if $\mathcal{M} \xrightarrow{\text{SNL}} \mathcal{N}$, then proceed as in case (2) except when $\mathcal{M} \xrightarrow{\text{SNA}} \mathcal{M}'$ changes the applicand of the subnode eliminated in $\mathcal{M} \xrightarrow{\text{SNL}} \mathcal{N}$, in which case let $\mathcal{N}' = \mathcal{N}$, and then $\mathcal{M}' \xrightarrow{\text{SNL}} \mathcal{N}'$ by the same transformation as $\mathcal{M} \xrightarrow{\text{SNL}} \mathcal{N}$;
4. if $\mathcal{M} \xrightarrow{\text{SNX}} \mathcal{N}$, then proceed as in case (2) except when $\mathcal{M} \xrightarrow{\text{SNX}} \mathcal{M}'$ expands a subnode a with respect to a subnode t (or a sink of t , which we will call by the same name) as a' and:
 - (a) $\mathcal{M} \xrightarrow{\text{SNL}} \mathcal{M}'$ eliminates t , in which case transform $\mathcal{N} \xrightarrow{\text{SNL}} \cdot \xrightarrow{\text{SNL}} \mathcal{N}'$ to eliminate both t and a' , and then $\mathcal{M}' \xrightarrow{\text{SNL}} \mathcal{N}'$ by eliminating a' ; or
 - (b) $\mathcal{M} \xrightarrow{\text{SNX}} \mathcal{M}'$ expands t as t' , in which case transform $\mathcal{N} \xrightarrow{\text{SNX}} \cdot \xrightarrow{\text{SNX}} \mathcal{N}'$ to expand t as t' as in $\mathcal{M} \xrightarrow{\text{SNX}} \mathcal{M}'$ and expand a' with respect to t' as a'' , and then $\mathcal{M}' \xrightarrow{\text{SNX}} \cdot \xrightarrow{\text{SNX}} \mathcal{N}'$ by expanding a with respect to t as a' and expanding a with respect to t' as a'' ; or

- (c) $\mathcal{M} \xrightarrow{\text{SNA}} \mathcal{M}'$ applies a to t , in which case transform $\mathcal{N} \xrightarrow{\text{SNA}} \cdot \xrightarrow{\text{SNL}} \mathcal{N}'$ to apply a' to t and eliminate a' , and then $\mathcal{M}' = \mathcal{N}'$; or
- (d) $\mathcal{M} \xrightarrow{\text{SMA}} \mathcal{M}'$ applies the submodel \mathcal{S} at a , in which case transform $\mathcal{N} \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ to apply the submodel \mathcal{S} at a and the submodel \mathcal{S}' of a' at a' and fully expand the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \mathbb{S}_{\mathcal{S}'} \sqcup \{a, a'\}$, and then $\mathcal{M}' \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ by fully expanding the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \{a\}$; or
- (e) $\mathcal{M} \xrightarrow{\text{SMA}} \mathcal{M}'$ applies the submodel \mathcal{U} at t , in which case transform $\mathcal{N} \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ to apply the submodel \mathcal{U} at t and fully expand a' , and then $\mathcal{M}' \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ by fully expanding a ;
5. if $\mathcal{M} \xrightarrow{\text{SNA}} \mathcal{N}$, then proceed as in case (2) except when $\mathcal{M} \xrightarrow{\text{SNA}} \mathcal{N}$ applies a subnode a to a subnode t and:
- (a) $\mathcal{M} \xrightarrow{\text{SNL}} \mathcal{M}'$ eliminates t , in which case transform $\mathcal{N} \xrightarrow{\text{SNL}} \mathcal{N}'$ to eliminate t , and then $\mathcal{M}' = \mathcal{N}'$; or
- (b) $\mathcal{M} \xrightarrow{\text{SNX}} \mathcal{M}'$ expands a with respect to t as a' , in which case let $\mathcal{N}' = \mathcal{N}$, and then $\mathcal{M}' \xrightarrow{\text{SNA}} \cdot \xrightarrow{\text{SNL}} \mathcal{N}'$ by applying a' to t and eliminating a' ; or
- (c) $\mathcal{M} \xrightarrow{\text{SNX}} \mathcal{M}'$ expands t as t' , in which case transform $\mathcal{N} \xrightarrow{\text{SNX}} \mathcal{N}'$ to expand t , and then $\mathcal{M}' \xrightarrow{\text{SNA}} \cdot \xrightarrow{\text{SNA}} \mathcal{N}'$ by applying a to t and applying a to t' ; or
- (d) $\mathcal{M} \xrightarrow{\text{SMA}} \mathcal{M}'$ applies the submodel \mathcal{S} at a , in which case transform $\mathcal{N} \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ to apply \mathcal{S} at a and apply the submodel \mathcal{S}' of t at t and fully expand the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \mathbb{S}_{\mathcal{S}'} \sqcup \{a, t\}$, and then $\mathcal{M}' \xrightarrow{\text{SNA}} \cdot \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ by applying a to t and applying the submodel \mathcal{S}'' of t at t and fully expanding the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \mathbb{S}_{\mathcal{S}''} \sqcup \{a, t\}$;
6. if $\mathcal{M} \xrightarrow{\text{SMA}} \mathcal{N}$, then proceed as in case (2) except when $\mathcal{M} \xrightarrow{\text{SMA}} \mathcal{N}$ applies a submodel \mathcal{S} at a subnode a and:
- (a) $\mathcal{M} \xrightarrow{\text{SNL}} \mathcal{M}'$ eliminates a , in which case transform $\mathcal{N} \xrightarrow{\text{SNL}} \cdots \xrightarrow{\text{SNL}} \mathcal{N}'$ to eliminate the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \{a\}$, and then $\mathcal{M}' = \mathcal{N}'$; or
- (b) $\mathcal{M} \xrightarrow{\text{SNX}} \mathcal{M}'$ expands a as a' , in which case transform $\mathcal{N} \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ to fully expand the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \{a\}$, and then $\mathcal{M}' \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ by applying \mathcal{S} at a and the submodel \mathcal{S}' of a' at a' and fully expanding the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \mathbb{S}_{\mathcal{S}'} \sqcup \{a, a'\}$; or
- (c) $\mathcal{M} \xrightarrow{\text{SNA}} \mathcal{M}'$ applies a to a subnode t , in which case transform $\mathcal{N} \xrightarrow{\text{SNA}} \cdot \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ to apply a to t and fully expand the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \{a, t\}$, and then $\mathcal{M}' \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SMA}} \cdot \xrightarrow{\text{SNX}} \cdots \xrightarrow{\text{SNX}} \mathcal{N}'$ by applying \mathcal{S} at a and applying the submodel \mathcal{S}' of t at t and fully expanding the subnodes $\mathbb{S}_{\mathcal{S}} \sqcup \mathbb{S}_{\mathcal{S}'} \sqcup \{a, t\}$.

The above argument could be formalized by an exhaustive analysis for each of the above cases demonstrating that the given transformation sequences yield the results claimed for any possible model. The RNX transformation can be handled with no particular difficulty: its treatment is similar to that of both SNX and SNA. Intuitively, the above argument shows why the normalization transformations in §3.3.1 constitute a consistent functional interpretation of MDPL models. The next section shows that this interpretation can be practically implemented.

3.3.3 Termination of Normal Form Reduction

Clearly there exist models to which infinite sequences of normalization transformations can be applied: for instance, a model with at least one applicand equal to a subnode admits infinitely many applications of the SNX transformation. Here we give a deterministic algorithm for applying the normalization transformations to an arbitrary model \mathcal{M} that will produce a normal form of \mathcal{M} if one exists. The process of applying this algorithm to a model \mathcal{M} is called *normal order reduction*.

The algorithm for normal order reduction of \mathcal{M} consists of applying the following procedure repeatedly until the result is a normal form:

1. If there is a subnode $a \in \mathbb{S}_{\mathcal{M}}$ to which SNL can be applied, let $\mathcal{M} \leftarrow \tau_a^{\text{SNL}}(\mathcal{M})$ and return \mathcal{M} .
2. If there is a subnode $t \in \mathbb{S}_{\mathcal{M}}$ to which SNA can be applied, let $\mathcal{M} \leftarrow \tau_t^{\text{SNA}}(\mathcal{M})$ and return \mathcal{M} .
3. If there is a subnode $a \in \mathbb{S}_{\mathcal{M}}$ such that there are multiple subnodes $t \in \mathbb{S}_{\mathcal{M}}$ or sinks $\mathcal{T} \in \mathbb{Y}_{\mathcal{M}}$ with $\pi_{\mathcal{M}}(t) = a$ or $\rho_{\mathcal{M}}(\mathcal{T}) = a$, respectively, let $\mathcal{M} \leftarrow \tau_t^{\text{SNX}}(\mathcal{M})$ or $\mathcal{M} \leftarrow \tau_{\mathcal{T}}^{\text{SNX}}(\mathcal{M})$, respectively, and return \mathcal{M} .
4. If there is a subnode $a \in \mathbb{S}_{\mathcal{M}}$ to which SMA can be applied, let $\mathcal{M} \leftarrow \tau_a^{\text{SMA}}(\mathcal{M})$ and return \mathcal{M} .
5. If there is a node $t \in \mathbb{U}_{\mathcal{M}}$ or sink $\mathcal{T} \in \mathbb{Y}_{\mathcal{M}}$ to which RNX can be applied, let $\mathcal{M} \leftarrow \tau_t^{\text{RNX}}(\mathcal{M})$ or $\mathcal{M} \leftarrow \tau_{\mathcal{T}}^{\text{RNX}}(\mathcal{M})$, respectively, and return \mathcal{M} .
6. If \mathcal{M} has a subnode $s \in \mathbb{S}_{\mathcal{M}}$ with a submodel applicand $\pi_{\mathcal{M}}(s) = \mathcal{Q} \in \mathcal{Q}_{\mathcal{M}}$, apply the above steps once to \mathcal{Q} and return the resulting \mathcal{M} .

The argument for termination of normal order reduction runs as follows. To show that the algorithm produces a normal form when one exists, we must show that the algorithm does not proceed for an infinite number of steps when a finite number of steps would reach a normal form, that is, when \mathcal{M} is normalizing. Clearly neither SNL nor SNA can be applied for an infinite number of steps. The SNX transformation is only applied until all subnodes are singular applicands or arguments, hence also cannot be applied for an infinite number of steps. The SMA transformation applied to a subnode could introduce a non-normalizing model as the applicand of a subnode of \mathcal{M} , but if \mathcal{M}

is normalizing, the subnode in question will be removed by the SNL transformation. Similarly, the RNX transformation can be applied for an infinite number of steps, but if \mathcal{M} is normalizing, there will be some step at which all nodes to which it could be applied have been removed by SNL (and all sinks to which it could be applied it has already been applied). Normal order reduction forms the theoretical basis of implementations of MDPL, though more intelligent procedures are required to achieve efficient reductions (§6.2).

3.4 Universal Computation

We now show that MDPL is universal (in the Church-Turing sense) by demonstrating an implementation of combinatory logic in MDPL. Since the sk combinator calculus is universal [15], an implementation of sk calculus in MDPL establishes the universality of MDPL. Such an implementation requires a procedure for constructing an MDPL model corresponding to an arbitrary sk -expression, and a procedure for constructing the normal form of the sk -expression from the normal form reduction of the MDPL model, neither procedure requiring universal computation.

3.4.1 Translation from sk Combinator Calculus to MDPL

We assume without loss of generality that the primitives \mathbb{P} in the formulation of sk correspond directly with primitives \mathbb{P} in the formulation of MDPL. Otherwise, the translation can be augmented by a mapping from primitives in sk to primitives in MDPL. The grammar of the sk calculus with primitives \mathbb{P} can be written as

$$\mathcal{G}_{sk} = [\{e\}, \mathbb{P} \sqcup \{s, k, (\cdot)\}, \{e \mapsto \mathbb{P}, e \mapsto s \mid k \mid (ee)\}, e].$$

Clearly we can construct a translator from \mathcal{G}_{sk} to the grammar of MDPL by defining a translation function \mathcal{L} on the four statement patterns p , k , s , (FE) , where the first is actually a statement pattern schema over all $p \in \mathbb{P}$, and the last has schematic variables F and E . The translation of (FE) of course depends recursively on the translations $\mathcal{L}(F)$ and $\mathcal{L}(E)$. Since this translation operates directly on the right-hand sides of the production rules in the context-free grammar \mathcal{G}_{sk} , it can be carried out by a pushdown automaton, hence respects the limits on the computational power of the construction.

We define \mathcal{L} using the visual representation for MDPL, representing the translation $\mathcal{L}(T)$ of a statement pattern T in \mathcal{G}_{sk} by a thick rectangle labeled by T , as shown. The visual representation of the model $\mathcal{L}(T)$ is understood to be superimposed on this rectangle wherever it appears, in the sense of §3.2. The translation function \mathcal{L} is then defined by the declarations represented in Figure 3-2, where $p \in \mathbb{P}$.

T

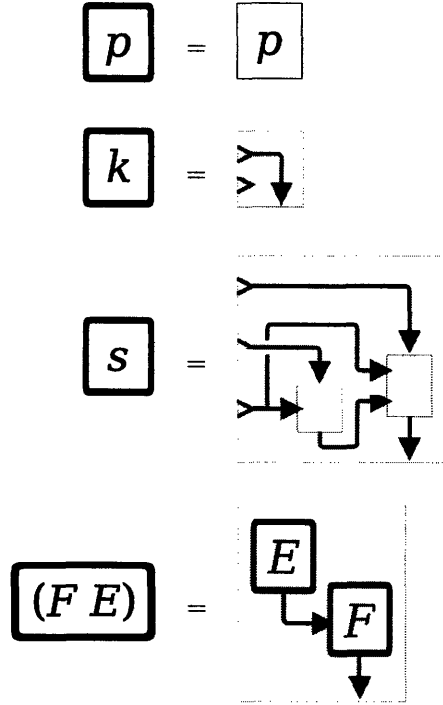


Figure 3-2: The translation function \mathcal{L} from the sk combinator calculus to MDPL

3.4.2 Correspondence of Normal Form Reduction in sk and MDPL

We now show that the translation of a statement in sk calculus and the translation of its normal form have essentially the same normal form in MDPL. We verify this by showing that the action of the two normal form reduction rules in sk calculus is essentially emulated under translation by the normalization transformations in MDPL.

For convenience, we define $\tau_{\dots}^{XYZ}(\mathcal{N}) = \mathcal{N}$ when the transformation XYZ cannot be applied to \mathcal{N} . Note from the form of the translations in Figure 3-2 and the normalization transformations that no internal model in the reduction of a translation of an sk -expression will ever have a node or sink with applicand or argument (respectively) equal to its root node, hence the RNX transformation is irrelevant to our discussion here (see §3.4.3).

k Reduction The k reduction rule in sk calculus can be written $((k X) Y) \xrightarrow{k} X$ for statement patterns X, Y . Figure 3-3 shows a partial normal form reduction of the translation of $((k X) Y)$ in MDPL to the model $\tilde{\mathcal{N}}$ second from the right. The full reduction cannot be obtained without a definition for $\mathcal{N} = \mathcal{L}(X)$ on the far right. However, if $\pi_{\mathcal{N}}(\tau_{\mathcal{N}}) = t \in \mathbb{S}_{\mathcal{N}}$, we can reduce $\tilde{\mathcal{N}}$ and \mathcal{N} to identical models as follows:

1. Reduce $\tilde{\mathcal{N}} \rightarrow \tilde{\mathcal{N}}'$ by applying SMA to its only subnode: $\tilde{\mathcal{N}}' = \tau_{\tilde{\mathcal{N}}}^{\text{SMA}}(\tilde{\mathcal{N}})$.

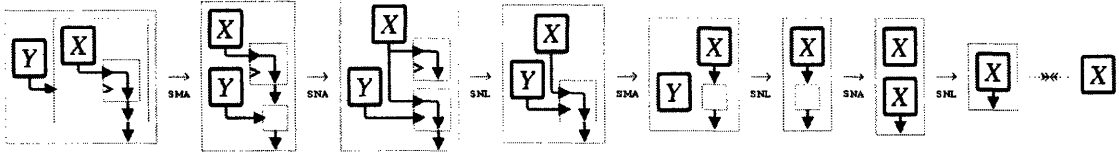


Figure 3-3: Normalization of the MDPL translation of $((k X) Y)$

2. Reduce $\tilde{\mathcal{N}}' \rightarrow \tilde{\mathcal{N}}''$ by applying SMA to t and its applicand: $\tilde{\mathcal{N}}'' = \tau_{\tilde{\mathcal{N}}', t}^{\text{SMA}}(\tilde{\mathcal{N}}')$.
3. Reduce $\tilde{\mathcal{N}}'' \rightarrow \tilde{\mathcal{N}}'''$ by applying SNL to the former applicand $a = \pi_{\tilde{\mathcal{N}}'}(t)$: $\tilde{\mathcal{N}}''' = \tau_{\tilde{\mathcal{N}}'', a}^{\text{SNL}}(\tilde{\mathcal{N}}'')$.
Then $\tilde{\mathcal{N}}''' = \mathcal{N}$.

If $\pi_{\mathcal{N}}(r_{\mathcal{N}}) \in \mathbb{X}_{\mathcal{N}}$, the reduction in step 2 (and following) is not applicable, and if $\pi_{\mathcal{N}}(r_{\mathcal{N}}) = \emptyset$, the reduction in step 1 (and following) is not applicable, and $\tilde{\mathcal{N}}$ and \mathcal{N} are not strictly equivalent under normalization. However, they are still *extensionally* equivalent under normalization: that is, substituting one for the other as a submodel of an internal model of some model \mathcal{M} does not change the normal form of \mathcal{M} . This relation is indicated by the dashed \dashrightarrow symbol in Figure 3-3. It can be formally verified by explicitly constructing the relevant reduction procedures as we have done for the case above. Intuitively, since $m_{\tilde{\mathcal{N}}} = n_{\tilde{\mathcal{N}}} = 0$, the sources and sinks of $\tilde{\mathcal{N}}$ will become corresponding sources and sinks of \mathcal{N} under SMA, and any sinks or nodes with applicand $\tilde{\mathcal{N}}$ will be assigned applicand \mathcal{N} under SMA. Since either SMA or SNL can eventually be applied to any subnodes with applicand $\tilde{\mathcal{N}}$ or \mathcal{N} , they will always lead to identical normal forms.

The above equivalence could be made strict by augmenting the set of normalization transformations while retaining their essential properties; however, as we shall see, only extensional equivalence is necessary to establish the implementation of sk in MDPL.

s Reduction The s reduction rule in sk calculus can be written $((s X) Y) Z \xrightarrow{s} ((X Z) (Y Z))$ for statement patterns X, Y, Z . Figure ?? shows partial normal form reductions of the translations of $((s X) Y) Z$ and $((X Z) (Y Z))$ in MDPL. The resulting normal forms in this case are identical.

To complete the implementation of sk calculus in MDPL, we require a procedure yielding the normal forms of sk -expressions from the normal form reductions of their translations under \mathcal{L} in MDPL. This is slightly more involved than the translation \mathcal{L} , since the MDPL normal form of the translation of an arbitrary sk -expression may not correspond directly to a right-hand side of one of the equations defining \mathcal{L} . However, it does have a characteristic form evident from the definition of \mathcal{L} and the normalization transformations: the root node has a subnode applicand and no sinks, subnodes have no sources and have subnode applicands, and sinks have arguments that are either subnodes or sources of the root node.

By essentially reversing the transformation carried out in the partial reduction of $((X Z) (Y Z))$ in the bottom of Figure ??, we can construct a model with the same normal form in which each

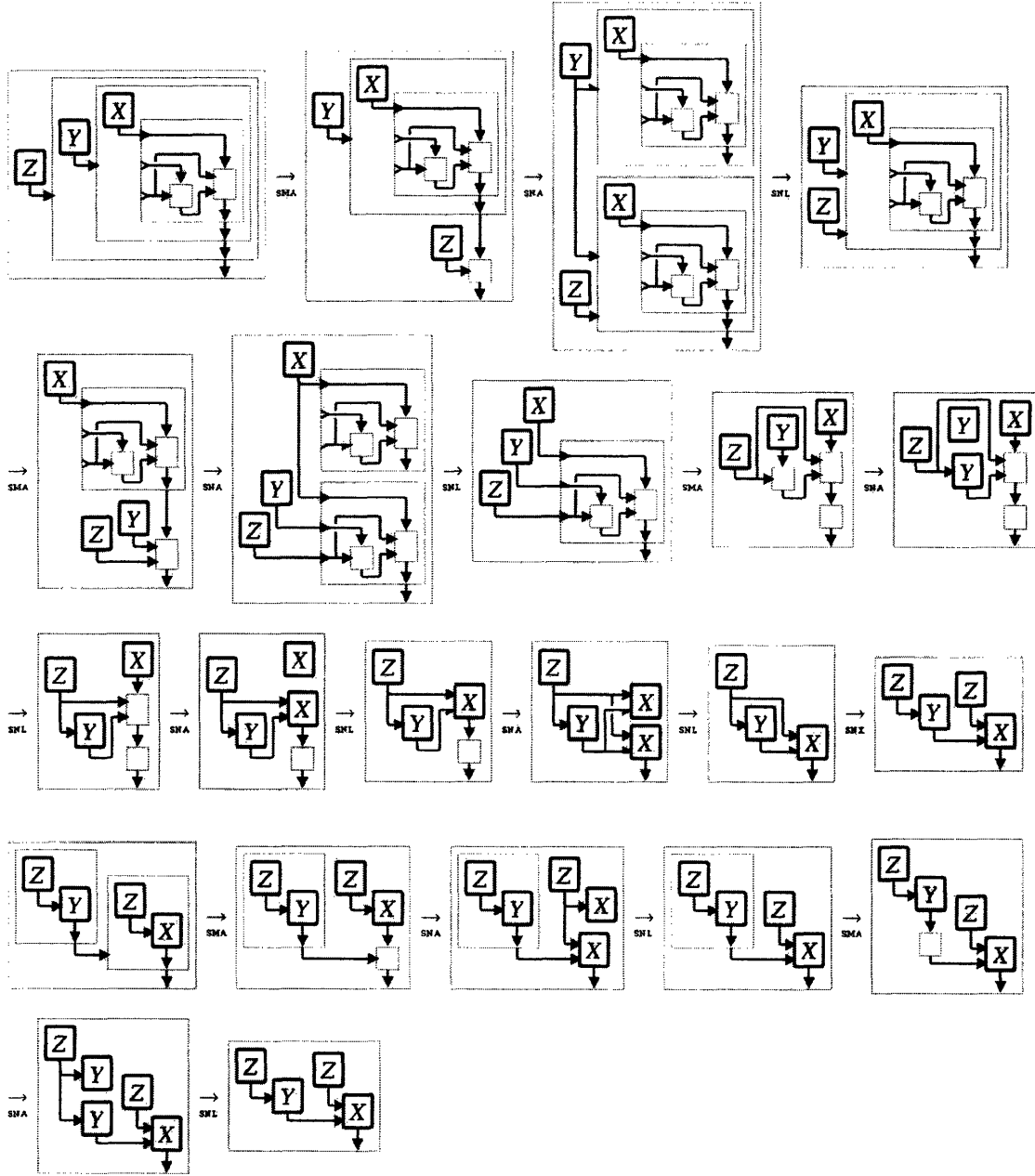


Figure 3-4: Normalizations of the MDPL translations of $((sX)Y)Z$ (top) and $((XZ)(YZ))$ (bottom).

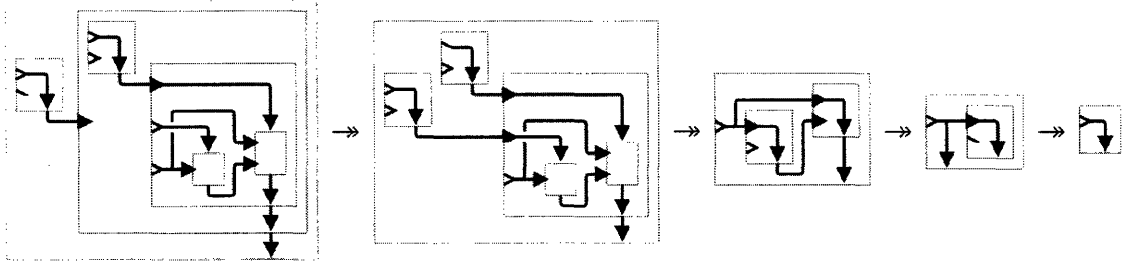


Figure 3-5: Abbreviated normal order reduction of the MDPL translation of $((s k) k)$.

internal model either matches the translation of the statement pattern $(F E)$, or is the normal form of the translation of a minimal normal-form sk -expression (one that does not contain a normal-form sk -expression as a subexpression). The minimal normal-form sk -expressions involving only s and k are k , s , $(k k)$, $(k s)$, $(s k)$, $(s s)$, $((s k) k)$, and $((s k) s)$, to which we must add those involving our particular primitives \mathbb{P} . For example, Figure 3-5 shows the MDPL normal form of the translation of $((s k) k)$, which can be interpreted as the identity function. By replacing models matching the translation of $(F E)$ by $(F E)$ and recursively replacing the normal forms of translations of minimal normal-form sk -expressions with said sk -expressions, we obtain the normal form of the original sk -expression with the normal form of whose translation we began.

It should now be clear why merely extensional equivalence of k reduction under translation in MDPL suffices for our implementation: When the result of the partial reduction in Figure 3-3 appears as a submodel, it yields an identical normal form as the direct translation of the normal form of the sk -expression. Hence any model that is an MDPL normal form will be identical to the normal form of the translation of the normal form of the sk -expression, but for the model itself (irrespective of its submodels). The latter case can be trivially recognized as an extension of the above procedure for recovering the normal-form sk -expression we desire.

3.4.3 Recursive Functions in MDPL

As noted above, the RNX transformation plays no role in the above implementation of sk calculus, since no internal models in the implementation have nodes or sinks with applicands or arguments (respectively) equal to their root nodes. Though we have seen in §3.2 that this is the most direct expression of functional recursion in MDPL, it is not necessary for expressing general recursive functions, since there are universal fixpoint models in MDPL analogous to the fixpoint combinators in the λ and sk calculi. Figure 3-6 shows one such fixpoint model (highlighted): a model (top) that applies it to an arbitrary submodel S has the same normal form as a second model (bottom) that applies S to the first model. Note this fixpoint model is substantially simpler than the direct translation of the simplest fixpoint model $((s k) (s (k ((s s) (s ((s s) k)))))) k)$ in sk calculus [66].

The existence of a fixpoint model means that for any model in MDPL, there is another model

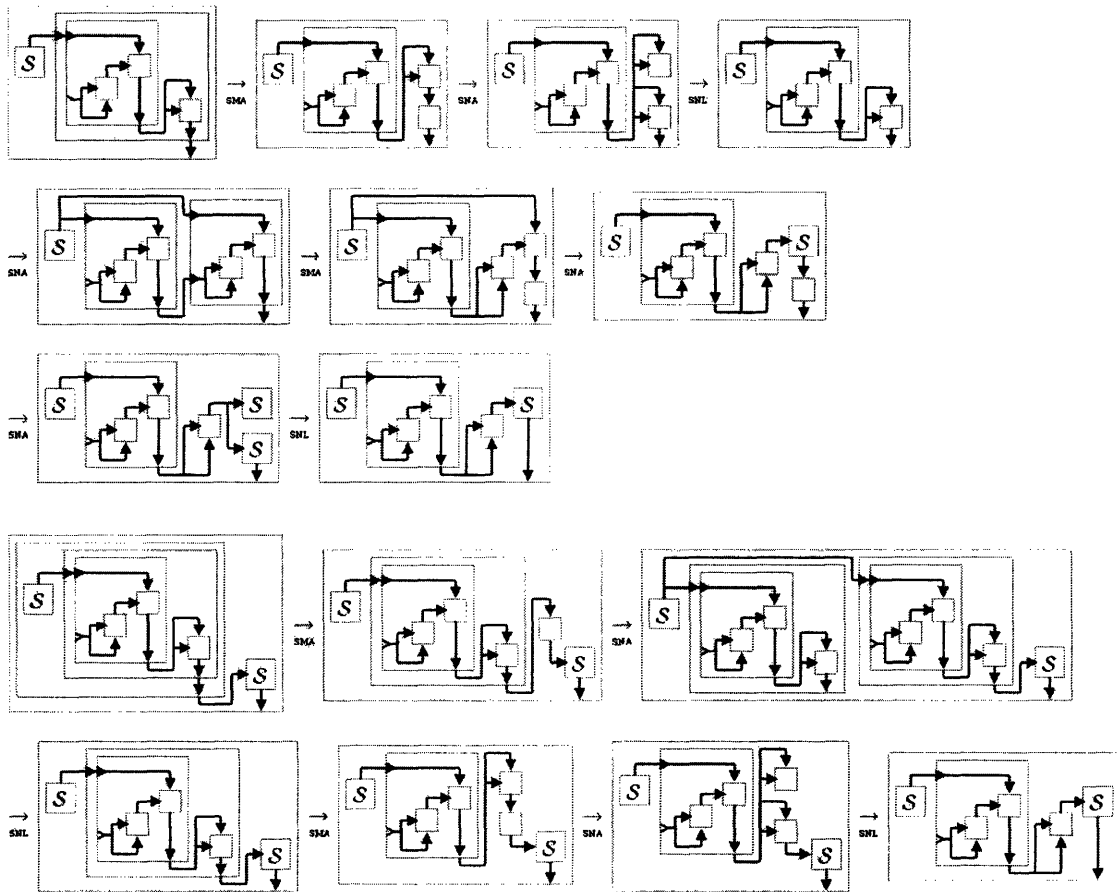


Figure 3-6: Reductions involving a fixpoint model (highlighted) applied to an arbitrary model S .

\mathcal{M} with the same normal form that employs no explicit recursion: $\pi_S(\mathbb{U}_S) \sqcup \{r_S\}$ for every internal model $S \in \mathcal{R}_{\mathcal{M}}$. Hence the MDPL language could be simplified without loss of computational power by restricting $\pi_{\mathcal{M}} : \mathbb{U}_{\mathcal{M}} \longrightarrow \mathbb{P} \sqcup \mathcal{Q}_{\mathcal{M}} \sqcup \mathbb{S}_{\mathcal{M}} \sqcup \mathbb{X}_{\mathcal{M}} \sqcup \{\emptyset\}$ and $\rho_{\mathcal{M}} : \mathbb{Y}_{\mathcal{M}} \longrightarrow \mathbb{P} \sqcup \mathbb{S}_{\mathcal{M}} \sqcup \mathbb{X}_{\mathcal{M}} \sqcup \{\emptyset\}$ in the specification of §3.1, and removing RNX from the normalization transformations of §3.3.1. However, this would require more cumbersome definitions for most practical recursive functions.

In this chapter, we have laid the theoretical foundations for the MDPL formalism and its consistent, practical interpretation as a model for all computable functions. The formal encoding of MDPL substantially generalizes those of related models of computation such as the lambda calculus and combinatory logic, but preserves some of the same functional ideas. However, the expressivity of the MDPL formalism makes the visual representation of §3.2 essential in performing intuitive operations on MDPL models, as we have seen in the figures above.

Chapter 4

Model Transformations

In Chapter 3, we saw that the MDPL language and its functional interpretation can consistently describe and execute all computable functions. While important, this property does not distinguish MDPL from most common programming languages, and says nothing about its practical facilities. In this chapter, we describe a number of transformations on MDPL models that have practical utility for developing programs expressed in the language. When implemented through an appropriate interface (Chapter 5), these transformations replace and supercede many of the functions grafted onto existing programming languages through extensions and development environments.

The next three sections describe three different types of transformations. The *semantic* transformations (§4.1) are used to change the functional interpretation of a model, and generally alter its normal form. The *syntactic* transformations (§4.2) are used to change the apparent structure of a model without altering its normal form. Finally, the *partial inversion* transformation (§4.3) changes the functional interpretation of a model and its normal form, but preserves the mathematical relations between each element of the model.

We define the *submodel graph* $\mathfrak{S}_{\mathcal{M}}$ of a model \mathcal{M} as the bipartite DAG on all internal models and nodes $\{\mathcal{R}_{\mathcal{M}}, \bigsqcup_{S \in \mathcal{R}_{\mathcal{M}}} \mathcal{U}_S\}$ of \mathcal{M} with edges such that

- nodes are children of the models in which they appear, $\forall S \in \mathcal{R}_{\mathcal{M}} \forall s \in \mathcal{U}_S N_{\mathfrak{S}_{\mathcal{M}}}^-(s) = \{S\}$
- (sub)nodes are parents of their submodel applicands, $\forall S \in \mathcal{R}_{\mathcal{M}} \forall s \in \mathcal{U}_S N_{\mathfrak{S}_{\mathcal{M}}}^+(s) = \{\pi_S(s)\} \cap \mathcal{Q}_S$.

The submodel graph reflects the hierarchical nesting of submodels within subnodes. In the visual representation of \mathcal{M} , we have $s \prec_{\mathfrak{S}_{\mathcal{M}}} t$ when the representation of t is somewhere inside the representation of s .

We define the *node graph* $\mathfrak{N}_{\mathcal{M}}$ of a model \mathcal{M} as the DAG on all nodes, sources, sinks, and primitives $\bigsqcup_{S \in \mathcal{R}_{\mathcal{M}}} \mathcal{U}_S \sqcup \mathcal{X}_S \sqcup \mathcal{Y}_S \sqcup \mathbb{P}$ of internal models of \mathcal{M} , with edges such that

- nodes are children of their applicands or root nodes of submodel applicands, and subnodes are

children of their sinks, $\forall_{S \in \mathcal{R}_M} \forall_{s \in \mathbb{U}_S} N_{\mathfrak{N}_M}^-(s) = \{\pi_S(s) \mid \pi_S(s) \in \mathbb{U}_S \sqcup \mathbb{X}_S \sqcup \mathbb{P}\} \sqcup \{r_Q \mid \pi_S(s) = Q \in \mathcal{Q}_S\} \sqcup \{\mathcal{Y} \in \mathbb{Y}_S \mid \psi_S(\mathcal{Y}) = s \in \mathbb{S}_S\}$

- sources of subnodes are children of their subnodes and of corresponding sinks of root nodes of submodel applicands of their subnodes, $\forall_{S \in \mathcal{R}_M} \forall_{s \in \mathbb{S}_S} \forall_{k \in \langle \nu_S(s) \rangle} N_{\mathfrak{N}_M}^-(s, k) = \{s\} \sqcup \{[r_Q, -k] \mid \pi_S(s) = Q \in \mathcal{Q}_S \wedge k \in \langle \mu_Q(r_Q) \rangle\}$
- sources of root nodes are children of corresponding sinks of subnodes to which their model is applicand, $\forall_{S \in \mathcal{R}_M} \forall_{k \in \langle m_S \rangle} N_{\mathfrak{N}_M}^-(r_S, k) = \bigsqcup_{\mathcal{R} \in \mathcal{R}_M} \{[s, -k] \mid \pi_{\mathcal{R}}(s) = S \wedge k \in \langle \mu_{\mathcal{R}}(s) \rangle\}_{s \in \mathbb{U}_{\mathcal{R}}}$
- sinks are children of their arguments, $\forall_{S \in \mathcal{R}_M} \forall_{\mathcal{Y} \in \mathbb{Y}_S} N_{\mathfrak{N}_M}^-(\mathcal{Y}) = \{\rho_S(\mathcal{Y}) \mid \rho_S(\mathcal{Y}) \in \mathbb{U}_S \sqcup \mathbb{X}_S \sqcup \mathbb{P}\}$.

The node graph \mathfrak{N}_M is a refinement of the link graphs $\{\mathfrak{H}_{\mathcal{R}}\}_{\mathcal{R} \in \mathcal{R}_M}$ that reflects structural dependencies in evaluation among the nodes, sources, and sinks of \mathcal{M} and its internal models. In the visual representation of \mathcal{M} , the edges of \mathfrak{N}_M are reflected by arrows, junctions of arrows to top or bottom sides of rectangles, and junctions of arrows to each another across left and right sides of rectangles.

We define the *restricted node graph* \mathfrak{N}_M of \mathcal{M} like \mathfrak{N}_M , except that some edges are removed so that

- subnodes are not children of their sinks, $\forall_{S \in \mathcal{R}_M} \forall_{s \in \mathbb{U}_S} N_{\mathfrak{N}_M}^-(s) = N_{\mathfrak{N}_M}^-(s) \setminus \{\mathcal{Y} \in \mathbb{Y}_S \mid \psi_S(\mathcal{Y}) = s \in \mathbb{S}_S\}$
- sources of subnodes are not children of their subnodes, $\forall_{S \in \mathcal{R}_M} \forall_{s \in \mathbb{S}_S} \forall_{k \in \langle \nu_S(s) \rangle} N_{\mathfrak{N}_M}^-(s, k) = N_{\mathfrak{N}_M}^-(s, k) \setminus \{s\}$.

We define the *applicand graph* \mathfrak{A}_M of a model \mathcal{M} as the digraph on the nodes $\bigsqcup_{S \in \mathcal{R}_M} \mathbb{U}_S$ of all internal models of \mathcal{M} whose edges join each node to the nodes succeeding it via only sources and sinks in \mathfrak{N}_M , $\forall_{S \in \mathcal{R}_M} \forall_{s \in \mathbb{U}_S} N_{\mathfrak{A}_M}^+(s) = \{u \in \mathbb{V}_{\mathcal{M}, s} \mid \exists_{t \in \mathbb{V}_{\mathcal{M}, s}} u \in \mathbb{V}_{\mathcal{M}, t}\}$ where $\mathbb{V}_{\mathcal{M}, s} = N_{\mathfrak{N}_M}^{\infty+}(s) \cap \bigsqcup_{S \in \mathcal{R}_M} \mathbb{U}_S$. The applicand graph reflects the inheritance of arguments among nodes: if the argument structure of one node is changed, this change must be reflected in any nodes succeeding it in the applicand graph to preserve the relationships of arguments in the functional interpretation of these nodes. In the visual representation of \mathcal{M} , the edges of \mathfrak{A}_M are reflected by continuous paths connecting top or bottom sides of rectangles to each other, made by arrows and junctions of arrows across left and right sides of rectangles.

For a node $s \in \mathbb{U}_M$, we define $\mathbb{I}_{\mathcal{M}, s} \equiv \{s\} \sqcup \{\mathcal{X} \in \mathbb{X}_M \mid \chi_M(\mathcal{X}) = s\}$ to consist of the node and its corresponding sources, and $\mathbb{J}_{\mathcal{M}, s} \equiv \{s\} \sqcup \{\mathcal{Y} \in \mathbb{Y}_M \mid \psi_M(\mathcal{Y}) = s\}$ to consist of the node and its corresponding sinks. In the visual representation of \mathcal{M} , there is an arrow from somewhere on the rectangle of $s \in \mathbb{U}_M$ to somewhere on the rectangle of $t \in \mathbb{U}_M$ when $N_{\mathfrak{N}_M}^+(\mathbb{I}_{\mathcal{M}, s}) \cap \mathbb{J}_{\mathcal{M}, t} \neq \emptyset$. We define $\underline{\mathbb{I}}_{\mathcal{M}, s} \equiv \mathbb{I}_{\mathcal{M}, s} \setminus \{s\}$ and $\underline{\mathbb{J}}_{\mathcal{M}, s} \equiv \mathbb{J}_{\mathcal{M}, s} \setminus \{s\}$.

4.1 Semantic Transformations

The semantic transformations involve reassigning or removing elements of a model that generally alter the model's normal form, and hence its functional interpretation. These transformations can be used to build up models from the *null model* \mathcal{Z} , the simplest possible model, defined by $\mathcal{Q}_{\mathcal{Z}} = \mathbb{S}_{\mathcal{Z}} = \emptyset$ and $m_{\mathcal{Z}} = n_{\mathcal{Z}} = 0$, whose visual representation is an empty rectangle; or to destroy part or all of any model.

As in §3.3.1, for each of the following transformations XYZ, we define a model \mathcal{M}' based on a given model \mathcal{M} satisfying certain conditions. For each case, when \mathcal{M} is an internal model of \mathcal{N} , we declare $\mathcal{N} \xrightarrow{\text{XYZ}} \mathcal{N}'$ where \mathcal{N}' is the model obtained from \mathcal{N} by replacing a particular instance of \mathcal{M} by \mathcal{M}' .

Subnode Deletion (DLS) The model \mathcal{M} has a subnode $a \in \mathbb{S}_{\mathcal{M}}$. Then the model \mathcal{M}' in $\mathcal{N}' = \nu_a^{\text{DLS}}(\mathcal{N})$ is constructed by removing the subnode a from \mathcal{M} and undefining any applicands and arguments equal to a or sources of a ,

$$\begin{aligned} \mathcal{Q}_{\mathcal{M}'} &= \pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}} \setminus \{a\}) \cap \mathcal{Q}_{\mathcal{M}} \\ r_{\mathcal{M}'} &= r_{\mathcal{M}} \\ \mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \setminus \{a\} \\ \mu_{\mathcal{M}'}(s) &= \mu_{\mathcal{M}}(s) \\ \nu_{\mathcal{M}'}(s) &= \nu_{\mathcal{M}}(s) \\ \pi_{\mathcal{M}'}(s) &= \begin{cases} \emptyset & \pi_{\mathcal{M}}(s) \in \mathbb{I}_{\mathcal{M},a} \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \rho_{\mathcal{M}'}(\mathcal{Y}) &= \begin{cases} \emptyset & \rho_{\mathcal{M}}(\mathcal{Y}) \in \mathbb{I}_{\mathcal{M},a} \\ \rho_{\mathcal{M}}(\mathcal{Y}) & \text{otherwise} \end{cases}, \end{aligned}$$

leaving the remaining nodes unchanged. This transformation effectively removes the subnode a and leaves its results undefined. In the visual representation, the rectangle of a and its contents are removed, along with any arrows to or from it.

Source Reassignment (RSX) The model \mathcal{M} has a node $a \in \mathbb{U}_{\mathcal{M}}$ with coarity $\nu_{\mathcal{M}}(a) = n$. Then the model \mathcal{M}' in $\mathcal{N}' = \nu_{a,i,j}^{\text{RSX}}(\mathcal{N})$ for $i, j \in \mathbb{N}$ is constructed by adding, removing, or reordering sources of a and the arguments or applicands equal to them so that the i th source of a becomes the

j th source of a , where a new source is added when $i > n$, and a source is removed when $i \leq n < j$,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= \mathcal{Q}_{\mathcal{M}} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \\
\mu_{\mathcal{M}'}(s) &= \mu_{\mathcal{M}}(s) \\
\nu_{\mathcal{M}'}(s) &= \begin{cases} n + \max\{1, j - n\} & i > n \\ n - 1 & i \leq n < j \\ n & i, j \leq n \end{cases} \\
\pi_{\mathcal{M}'}(s) &= \delta_{\mathcal{M}}(\pi_{\mathcal{M}}(s), a, i, j) \\
\rho_{\mathcal{M}'}(\mathcal{Y}) &= \delta_{\mathcal{M}}(\rho_{\mathcal{M}}(\mathcal{Y}), a, i, j),
\end{aligned}$$

where

$$\delta_{\mathcal{M}}(u, a, i, j) \equiv \begin{cases} \begin{cases} [a, l] & l < j \\ [a, l + 1] & l \geq j \end{cases} & i > n \\ \begin{cases} [a, l] & l < i \\ \emptyset & l = i \\ [a, l - 1] & l > i \end{cases} & i \leq n < j \\ \begin{cases} [a, l] & l \neq i, j \\ [a, j] & l = i \\ [a, i] & l = j \end{cases} & i, j \leq n \\ u & \text{otherwise} \end{cases} \quad u = [a, l] \in \mathbb{I}_{\mathcal{M}, a}$$

with $n = \nu_{\mathcal{M}}(a)$. This transformation effectively inserts, deletes, or permutes one source of a without changing the external connections to existing sources of a that remain. In the visual representation, one bracket on the right side of the rectangle of a is added, removed, or moved to a different position on the side of the rectangle along with the tail of any arrows out of it.

Sink Reassignment (RSY) The model \mathcal{M} has a node $a \in \mathbb{U}_{\mathcal{M}}$ with arity $\mu_{\mathcal{M}}(a) = m$. Then the model \mathcal{M}' in $\mathcal{N}' = \nu_{a, i, j}^{\text{RSY}}(\mathcal{N})$ for $i, j \in \mathbb{N}$ is constructed by adding, removing, or reordering sinks of a and their arguments so that the i th sink of a becomes the j th sink of a , where a new sink is added

when $i > m$, and a sink is removed when $i \leq m < j$,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= \mathcal{Q}_{\mathcal{M}} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}} \\
\mathcal{S}_{\mathcal{M}'} &= \mathcal{S}_{\mathcal{M}} \\
\mu_{\mathcal{M}'}(s) &= \begin{cases} \begin{cases} m + \max\{1, j - m\} & i > m \\ m - 1 & i \leq m < j \\ m & i, j \leq m \end{cases} & s = a \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}'}(s) &= \nu_{\mathcal{M}}(s) \\
\pi_{\mathcal{M}'}(s) &= \pi_{\mathcal{M}}(s) \\
\rho_{\mathcal{M}'}(\mathcal{Y}) &= \epsilon_{\mathcal{M}}(\mathcal{Y}, a, i, j),
\end{aligned}$$

where

$$\epsilon_{\mathcal{M}}(\mathcal{Y}, a, i, j) \equiv \begin{cases} \begin{cases} \bar{\rho}_{\mathcal{M}}(a, -l) & l < j \\ \emptyset & l = j \\ \rho_{\mathcal{M}}(a, -l+1) & l > j \end{cases} & i > m \\ \begin{cases} \rho_{\mathcal{M}}(a, -l) & l < j \\ \rho_{\mathcal{M}}(a, -l-1) & l \geq j \end{cases} & i \leq m < j \quad \mathcal{Y} = [a, -l] \in \mathbb{J}_{\mathcal{M}, a} \\ \begin{cases} \rho_{\mathcal{M}}(a, -l) & l \neq i, j \\ \rho_{\mathcal{M}}(a, -j) & l = i \\ \rho_{\mathcal{M}}(a, -i) & l = j \end{cases} & i, j \leq m \\ \rho_{\mathcal{M}}(\mathcal{Y}) & \text{otherwise} \end{cases}$$

with $m = \mu_{\mathcal{M}}(a)$. This transformation effectively inserts, deletes, or permutes one sink of a without changing the external connections to existing sinks of a that remain. In the visual representation, one bracket on the left side of the rectangle of a is added, removed, or moved to a different position on the side of the rectangle along with the head of any arrow into it.

Argument Reassignment (RSR) The model \mathcal{M} has instances of an internal model $\mathcal{A} \in \mathcal{R}_{\mathcal{M}}$ with a node or source $\varkappa \in \mathbb{I}_{\mathcal{A}, a}$ for some $a \in \mathbb{U}_{\mathcal{A}}$ and of an internal model $\mathcal{T} \in \mathcal{R}_{\mathcal{M}}$ with a sink $\mathcal{K} \in \mathbb{J}_{\mathcal{T}, t}$ for some $t \in \mathbb{U}_{\mathcal{T}}$, hence $\mathcal{M} \prec_{\mathfrak{S}_{\mathcal{M}}} a$ and $\mathcal{M} \prec_{\mathfrak{S}_{\mathcal{M}}} t$; there is no submodel $\mathcal{Q} \in \mathcal{Q}_{\mathcal{M}}$ for which $\mathcal{Q} \prec_{\mathfrak{S}_{\mathcal{M}}} a$ and $\mathcal{Q} \prec_{\mathfrak{S}_{\mathcal{M}}} t$; and we have $\bar{a} \not\prec_{\mathfrak{S}_{\mathcal{M}}} \bar{t}$ where $\bar{a}, \bar{t} \in \mathbb{U}_{\mathcal{M}}$ and $\bar{a} \prec_{\mathfrak{S}_{\mathcal{M}}} a$ and $\bar{t} \prec_{\mathfrak{S}_{\mathcal{M}}} t$. Alternatively, we may take \varkappa to be primitive, $\varkappa \in \mathbb{P}$, or undefined, $\varkappa = \emptyset$, with $\mathcal{A} = \mathcal{T} = \mathcal{M}$.

If $\mathcal{A} = \mathcal{M}$ and $\mathcal{T} = \mathcal{M}$, then the model \mathcal{M}' in $\mathcal{N}' = v_{\kappa, \mathcal{K}}^{\text{RSR}}(\mathcal{N})$ is constructed by assigning κ to be the argument of \mathcal{K} ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= \mathcal{Q}_{\mathcal{M}} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \\
\mu_{\mathcal{M}'}(s) &= \mu_{\mathcal{M}}(s) \\
\nu_{\mathcal{M}'}(s) &= \nu_{\mathcal{M}}(s) \\
\pi_{\mathcal{M}'}(s) &= \pi_{\mathcal{M}}(s) \\
\rho_{\mathcal{M}'}(\mathcal{Y}) &= \begin{cases} \kappa & \mathcal{Y} = \mathcal{K} \\ \rho_{\mathcal{M}}(\mathcal{Y}) & \text{otherwise} \end{cases},
\end{aligned}$$

leaving the other nodes unchanged; otherwise if $\mathcal{A} \neq \mathcal{M}$ and $\mathcal{T} = \mathcal{M}$, then the model \mathcal{M}' is constructed by replacing the submodel applicand $\bar{\mathcal{A}} = \pi_{\mathcal{M}}(\bar{a})$ of \bar{a} by a new submodel $\bar{\mathcal{A}}'$ applying the RSR transformation to κ and a new sink of $r_{\bar{\mathcal{A}}}$, and assigning a new corresponding source of \bar{a} to be the argument of \mathcal{K} ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= (\pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}} \setminus \{\bar{a}\}) \cap \mathcal{Q}_{\mathcal{M}}) \cup \{\bar{\mathcal{A}}'\} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \\
\mu_{\mathcal{M}'}(s) &= \mu_{\mathcal{M}}(s) \\
\nu_{\mathcal{M}'}(s) &= \begin{cases} \nu_{\mathcal{M}}(\bar{a}) + 1 & s = \bar{a} \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}'}(s) &= \begin{cases} \bar{\mathcal{A}}' & s = \bar{a} \\ \delta_{\mathcal{M}}(\pi_{\mathcal{M}}(s), \bar{a}, \nu_{\mathcal{M}}(\bar{a}) + 1, n_{\bar{\mathcal{A}}} + 1) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}'}(\mathcal{Y}) &= \begin{cases} [\bar{a}, n_{\bar{\mathcal{A}}} + 1] & \mathcal{Y} = \mathcal{K} \\ \delta_{\mathcal{M}}(\rho_{\mathcal{M}}(\mathcal{Y}), \bar{a}, \nu_{\mathcal{M}}(\bar{a}) + 1, n_{\bar{\mathcal{A}}} + 1) & \text{otherwise} \end{cases},
\end{aligned}$$

where $\bar{\mathcal{A}} = \pi_{\mathcal{M}}(\bar{a})$ and $\bar{\mathcal{A}}' = v_{\kappa, \mathcal{U}}^{\text{RSR}}(v_{r_{\bar{\mathcal{A}}}, n_{\bar{\mathcal{A}}}+1, n_{\bar{\mathcal{A}}}+1}^{\text{RSY}}(\bar{\mathcal{A}}))$ with $\mathcal{U} = [r_{\bar{\mathcal{A}}}, -n_{\bar{\mathcal{A}}} - 1] \in \mathbb{Y}_{\bar{\mathcal{A}}}$; otherwise if $\mathcal{T} \neq \mathcal{M}$, then the model $\mathcal{M}' = v_{\kappa, \mathcal{V}}^{\text{RSR}}(\mathcal{M}'')$ where $\mathcal{V} = [\bar{t}, -m_{\bar{\mathcal{T}}} - 1]$ and \mathcal{M}'' is constructed by applying the RSR transformation to κ and a new sink of \bar{t} in the model \mathcal{M}'' constructed by applying

the RSR transformation to a new corresponding source of the root node of \bar{t} and to \mathcal{K} ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}''} &= (\pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}} \setminus \{\bar{t}\}) \cap \mathcal{Q}_{\mathcal{M}}) \cup \{\bar{T}'\} \\
r_{\mathcal{M}''} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}''} &= \mathbb{S}_{\mathcal{M}} \\
\mu_{\mathcal{M}''}(s) &= \begin{cases} \mu_{\mathcal{M}}(\bar{t}) + 1 & s = \bar{t} \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}''}(s) &= \nu_{\mathcal{M}}(s) \\
\pi_{\mathcal{M}''}(s) &= \begin{cases} \bar{T}' & s = \bar{t} \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}''}(\mathcal{Y}) &= \epsilon_{\mathcal{M}}(\mathcal{Y}, \bar{t}, \mu_{\mathcal{M}}(\bar{t}) + 1, m_{\bar{T}} + 1),
\end{aligned}$$

where $\bar{T} = \pi_{\mathcal{M}}(\bar{t})$ and $\bar{T}' = v_{\mathcal{W}, \mathcal{K}}^{\text{RSR}}(v_{r_{\bar{T}}, m_{\bar{T}}+1, m_{\bar{T}}+1}^{\text{RSX}}(\bar{T}))$ with $\mathcal{W} = [r_{\bar{T}}, m_{\bar{T}} + 1]$. This transformation effectively assigns the argument of \mathcal{K} to be κ by adding new sources and sinks connected by arguments and applicands to the intervening internal models of \mathcal{M} . In the visual representation, a new continuous path is formed connecting κ to the bracket of \mathcal{K} , made by arrows and junctions of arrows across left and right sides of the rectangles containing κ or \mathcal{K} .

Applicand Reassignment (RSP) The model \mathcal{M} has instances of an internal model $\mathcal{A} \in \mathcal{R}_{\mathcal{M}}$ with a node or source $\kappa \in \mathbb{I}_{\mathcal{A}, a}$ for some $a \in \mathbb{U}_{\mathcal{A}}$ and of an internal model $\mathcal{T} \in \mathcal{R}_{\mathcal{M}}$ with a node $t \in \mathbb{U}_{\mathcal{T}}$, hence $\mathcal{M} \prec_{\mathfrak{S}_{\mathcal{M}}} a$ and $\mathcal{M} \prec_{\mathfrak{S}_{\mathcal{M}}} t$; there is no submodel $\mathcal{Q} \in \mathcal{Q}_{\mathcal{M}}$ for which $\mathcal{Q} \prec_{\mathfrak{S}_{\mathcal{M}}} a$ and $\mathcal{Q} \prec_{\mathfrak{S}_{\mathcal{M}}} t$; and we have $\bar{a} \not\prec_{\mathfrak{S}_{\mathcal{M}}} \bar{t}$ where $\bar{a}, \bar{t} \in \mathbb{U}_{\mathcal{M}}$ and $\bar{a} \prec_{\mathfrak{S}_{\mathcal{M}}} a$ and $\bar{t} \prec_{\mathfrak{S}_{\mathcal{M}}} t$. Alternatively, we may take κ to be primitive, $\kappa \in \mathbb{P}$, or undefined, $\kappa = \emptyset$, with $\mathcal{A} = \mathcal{T} = \mathcal{M}$.

If $\mathcal{A} = \mathcal{M}$ and $\mathcal{T} = \mathcal{M}$, then the model \mathcal{M}' in $\mathcal{N}' = v_{\kappa, \bar{t}}^{\text{RSP}}(\mathcal{N})$ is constructed by assigning κ to be the applicand of t ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= \pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}} \setminus \{t\}) \cap \mathcal{Q}_{\mathcal{M}} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \\
\mu_{\mathcal{M}'}(s) &= \mu_{\mathcal{M}}(s) \\
\nu_{\mathcal{M}'}(s) &= \nu_{\mathcal{M}}(s) \\
\pi_{\mathcal{M}'}(s) &= \begin{cases} \kappa & s = t \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}'}(\mathcal{Y}) &= \rho_{\mathcal{M}}(\mathcal{Y}),
\end{aligned}$$

leaving the other nodes unchanged; otherwise if $\mathcal{A} \neq \mathcal{M}$ and $\mathcal{T} = \mathcal{M}$, then the model \mathcal{M}' is constructed by replacing the submodel applicand $\bar{\mathcal{A}} = \pi_{\mathcal{M}}(\bar{a})$ of \bar{a} by a new submodel $\bar{\mathcal{A}}'$ applying the RSR transformation to \varkappa and a new sink of $r_{\bar{\mathcal{A}}}$, and assigning a new corresponding source of \bar{a} to be the applicand of t ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= (\pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}} \setminus \{\bar{a}, t\}) \cap \mathcal{Q}_{\mathcal{M}}) \cup \{\bar{\mathcal{A}}'\} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}'} &= \mathbb{S}_{\mathcal{M}} \\
\mu_{\mathcal{M}'}(s) &= \mu_{\mathcal{M}}(s) \\
\nu_{\mathcal{M}'}(s) &= \begin{cases} \nu_{\mathcal{M}}(\bar{a}) + 1 & s = \bar{a} \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}'}(s) &= \begin{cases} \bar{\mathcal{A}}' & s = \bar{a} \\ [\bar{a}, n_{\bar{\mathcal{A}}} + 1] & s = t \\ \delta_{\mathcal{M}}(\pi_{\mathcal{M}}(s), \bar{a}, \nu_{\mathcal{M}}(\bar{a}) + 1, n_{\bar{\mathcal{A}}} + 1) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}'}(\mathcal{Y}) &= \delta_{\mathcal{M}}(\rho_{\mathcal{M}}(\mathcal{Y}), \bar{a}, \nu_{\mathcal{M}}(\bar{a}) + 1, n_{\bar{\mathcal{A}}} + 1),
\end{aligned}$$

where $\bar{\mathcal{A}} = \pi_{\mathcal{M}}(\bar{a})$ and $\bar{\mathcal{A}}' = v_{\varkappa, \mathcal{U}}^{\text{RSR}}(v_{r_{\bar{\mathcal{A}}}, n_{\bar{\mathcal{A}}} + 1, n_{\bar{\mathcal{A}}} + 1}^{\text{RSY}}(\bar{\mathcal{A}}))$ with $\mathcal{U} = [r_{\bar{\mathcal{A}}}, -n_{\bar{\mathcal{A}}} - 1] \in \mathbb{J}_{\bar{\mathcal{A}}, r_{\bar{\mathcal{A}}}}$; otherwise if $\mathcal{T} \neq \mathcal{M}$, then the model $\mathcal{M}' = v_{\varkappa, \mathcal{V}}^{\text{RSR}}(\mathcal{M}'')$ where $\mathcal{V} = [\bar{t}, -m_{\bar{\mathcal{T}}} - 1]$ is constructed by applying the RSR transformation to \varkappa and a new sink of \bar{t} in the model \mathcal{M}'' constructed by applying the RSP transformation to a new corresponding source of the root node of \bar{t} and to t ,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}''} &= (\pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}} \setminus \{\bar{t}\}) \cap \mathcal{Q}_{\mathcal{M}}) \cup \{\bar{\mathcal{T}}'\} \\
r_{\mathcal{M}''} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}''} &= \mathbb{S}_{\mathcal{M}} \\
\mu_{\mathcal{M}''}(s) &= \begin{cases} \mu_{\mathcal{M}}(\bar{t}) + 1 & s = \bar{t} \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}''}(s) &= \nu_{\mathcal{M}}(s) \\
\pi_{\mathcal{M}''}(s) &= \begin{cases} \bar{\mathcal{T}}' & s = \bar{t} \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}''}(\mathcal{Y}) &= \epsilon_{\mathcal{M}}(\mathcal{Y}, \bar{t}, \mu_{\mathcal{M}}(\bar{t}) + 1, m_{\bar{\mathcal{T}}} + 1),
\end{aligned}$$

where $\bar{\mathcal{T}} = \pi_{\mathcal{M}}(\bar{t})$ and $\bar{\mathcal{T}}' = v_{\mathcal{W}, t}^{\text{RSP}}(v_{r_{\bar{\mathcal{T}}}, m_{\bar{\mathcal{T}}} + 1, m_{\bar{\mathcal{T}}} + 1}^{\text{RSX}}(\bar{\mathcal{T}}))$ with $\mathcal{W} = [r_{\bar{\mathcal{T}}}, m_{\bar{\mathcal{T}}} + 1]$. This transformation effectively assigns the applicand of t to be \varkappa by adding new sources and sinks connected by argu-

ments and applicands to the intervening internal models of \mathcal{M} . In the visual representation, a new continuous path is formed connecting \varkappa to the top or bottom side of the rectangle of t , made by arrows and junctions of arrows across left and right sides of the rectangles containing \varkappa or t .

4.2 Syntactic Transformations

The syntactic transformations involve rearranging elements of a model while preserving the model's normal form, and hence its functional interpretation. These transformations can be used to reorganize the elements of a model, often to facilitate meaningful changes to the model using the semantic transformations of §4.1.

As in the previous section, for each of the following transformations XYZ, we define a model \mathcal{M}' based on a given model \mathcal{M} satisfying certain preconditions. For each case, when \mathcal{M} is an internal model of \mathcal{N} , we declare $\mathcal{N} \xrightarrow{\text{XYZ}} \mathcal{N}'$ where \mathcal{N}' is the model obtained from \mathcal{N} by replacing a particular instance of \mathcal{M} by \mathcal{M}' . Herein we adopt the convention that a transformation XYZ leaves \mathcal{N} unchanged, $v_{\dots}^{\text{XYZ}}(\mathcal{N}) = \mathcal{N}$, when the preconditions of XYZ on \mathcal{M} do not hold.

Subnode Relocation (RLS) The model \mathcal{M} has instances of an internal model $\mathcal{A} \in \mathcal{R}_{\mathcal{M}}$ with a subnode $a \in \mathbb{S}_{\mathcal{A}}$ and of an internal model $\mathcal{T} \in \mathcal{R}_{\mathcal{M}}$, hence $\mathcal{M} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{A}$ and $\mathcal{M} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{T}$; there is no submodel $\mathcal{Q} \in \mathcal{Q}_{\mathcal{M}}$ for which $\mathcal{Q} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{A}$ and $\mathcal{Q} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{T}$; and we have that

- if any node of a model succeeding \mathcal{M} and preceding or equal to \mathcal{A} in the submodel graph of \mathcal{M} :

- precedes a in the node graph of \mathcal{M} , $\exists s \in \mathcal{R}_{\mathcal{M}} | \mathcal{M} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} s \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{A} \overset{\mathfrak{N}_{\mathcal{M}}}{\prec} s \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} a$, then we have either $\mathcal{T} = \mathcal{M}$ or $\bar{a} \not\prec \bar{t}$ where $\bar{a}, \bar{t} \in \mathbb{U}_{\mathcal{M}}$ and $\bar{a} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{A}$ and $\bar{t} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{T}$
- succeeds a in the node graph of \mathcal{M} , $\exists s \in \mathcal{R}_{\mathcal{M}} | \mathcal{M} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} s \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{A} \overset{\mathfrak{N}_{\mathcal{M}}}{\prec} s \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} a$, then we have either $\mathcal{T} = \mathcal{M}$ or $\bar{a} \not\prec \bar{t}$ where $\bar{a}, \bar{t} \in \mathbb{U}_{\mathcal{M}}$ and $\bar{a} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{A}$ and $\bar{t} \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{T}$

- any node of a model preceding \mathcal{T} in the submodel graph of \mathcal{M} that:

- succeeds a in the node graph of \mathcal{M} does not precede in the restricted link graph of its model any subnode preceding \mathcal{T} in the submodel graph of \mathcal{M} , $\forall s \in \mathcal{R}_{\mathcal{M}} | s \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{T} \overset{\mathfrak{N}_{\mathcal{M}}}{\prec} s \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} a \forall t \in \mathbb{S}_{\mathcal{S}} | t \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{T} \not\prec \underline{\mathcal{S}}_s$
 t
- precedes a in the node graph of \mathcal{M} does not succeed in the restricted link graph of its model any subnode preceding \mathcal{T} in the submodel graph of \mathcal{M} , $\forall s \in \mathcal{R}_{\mathcal{M}} | s \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{T} \overset{\mathfrak{N}_{\mathcal{M}}}{\prec} s \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} a \forall t \in \mathbb{S}_{\mathcal{S}} | t \underset{\mathfrak{E}_{\mathcal{M}}}{\prec} \mathcal{T} \not\prec \underline{\mathcal{S}}_s$
 t .

If $\mathcal{A} = \mathcal{M}$ and $\mathcal{T} = \mathcal{M}$, then the model \mathcal{M}' in $\mathcal{N}' = v_{a, \mathcal{T}}^{\text{RLS}}(\mathcal{N})$ is unchanged, $\mathcal{M}' = \mathcal{M}$.

Otherwise if $\mathcal{A} = \pi_{\mathcal{M}}(\bar{a})$ where $\bar{a} \in \mathbb{U}_{\mathcal{M}}$ and $\mathcal{T} = \mathcal{M}$, then \mathcal{M}' is constructed as follows: first form \mathcal{M}'' from \mathcal{M} by replacing $\mathcal{A} = \pi_{\mathcal{M}}(\bar{a})$ by a model $\bar{\mathcal{A}}$ with the same functional interpretation constructed by embedding \mathcal{A} as the submodel of a single subnode \bar{a} ,

$$\begin{aligned}
\mathcal{Q}_{\bar{\mathcal{A}}} &= \{\mathcal{A}\} \\
r_{\bar{\mathcal{A}}} &= r_{\mathcal{A}} \\
\mathcal{S}_{\bar{\mathcal{A}}} &= \{\bar{a}\} \\
\mu_{\bar{\mathcal{A}}}(s) &= \begin{cases} n_{\mathcal{A}} & s = r_{\bar{\mathcal{A}}} \\ m_{\mathcal{A}} & s = \bar{a} \end{cases} \\
\nu_{\bar{\mathcal{A}}}(s) &= \begin{cases} m_{\mathcal{A}} & s = r_{\bar{\mathcal{A}}} \\ n_{\mathcal{A}} & s = \bar{a} \end{cases} \\
\pi_{\bar{\mathcal{A}}}(s) &= \begin{cases} \begin{cases} \emptyset & \pi_{\mathcal{A}}(r_{\mathcal{A}}) = \emptyset \\ \bar{a} & \text{otherwise} \end{cases} & s = r_{\bar{\mathcal{A}}} \\ \mathcal{A} & s = \bar{a} \end{cases} \\
\rho_{\bar{\mathcal{A}}}(s, -k) &= \begin{cases} [\bar{a}, k] & s = r_{\bar{\mathcal{A}}} \\ [r_{\bar{\mathcal{A}}}, k] & s = \bar{a} \end{cases},
\end{aligned}$$

then construct \mathcal{M}''' from \mathcal{M}'' by replacing $\bar{\mathcal{A}}$ with the model $\bar{\mathcal{A}}'$ constructed by moving the subnode a out of the submodel \mathcal{A} as a second subnode of $\bar{\mathcal{A}}'$ and rearranging sources and sinks and reassigning

arguments and applicands to preserve the functional interpretation of $\bar{\mathcal{A}}$,

$$\begin{aligned}
\mathcal{Q}_{\bar{\mathcal{A}}'} &= \{\mathcal{A}'\} \cup (\{\pi_{\mathcal{A}}(a)\} \cap \mathcal{Q}_{\mathcal{A}}) \\
r_{\bar{\mathcal{A}}'} &= r_{\bar{\mathcal{A}}} \\
\mathbb{S}_{\bar{\mathcal{A}}'} &= \{a, \bar{a}\} \\
\mu_{\bar{\mathcal{A}}'}(s) &= \begin{cases} \max\{\mu_{\bar{\mathcal{A}}}(\bar{a}), m_{\mathcal{A}}\} - |\mathcal{U}^-| + |\mathcal{U}^+| & s = \bar{a} \\ \mu_{\mathcal{A}}(a) & s = a \\ \mu_{\bar{\mathcal{A}}}(s) & \text{otherwise} \end{cases} \\
\nu_{\bar{\mathcal{A}}'}(s) &= \begin{cases} \max\{\nu_{\bar{\mathcal{A}}}(\bar{a}), n_{\mathcal{A}}\} - |\mathcal{V}^-| + |\mathcal{V}^+| & s = \bar{a} \\ \nu_{\mathcal{A}}(a) & s = a \\ \nu_{\bar{\mathcal{A}}}(s) & \text{otherwise} \end{cases} \\
\pi_{\bar{\mathcal{A}}'}(s) &= \begin{cases} \mathcal{A}' & s = \bar{a} \\ \zeta_{\bar{\mathcal{A}}, \mathcal{A}}(\pi, s, a, \bar{a}, s) & \text{otherwise} \end{cases} \\
\rho_{\bar{\mathcal{A}}'}(s, -k) &= \begin{cases} \begin{cases} \bar{\rho}_{\bar{\mathcal{A}}}(\bar{a}, -k - \varpi(\mathcal{U}^-, k)) & k \in \langle m_{\mathcal{A}} - |\mathcal{U}^-| \rangle \\ \mathcal{U}_{k-m_{\mathcal{A}}+|\mathcal{U}^-|}^+ & k - m_{\mathcal{A}} + |\mathcal{U}^-| \in \langle |\mathcal{U}^+| \rangle \\ \bar{\rho}_{\bar{\mathcal{A}}}(\bar{a}, -k + |\mathcal{U}^-| - |\mathcal{U}^+|) & \text{otherwise} \end{cases} & s = \bar{a} \\ \zeta_{\bar{\mathcal{A}}, \mathcal{A}}(\rho, [s, -k], a, \bar{a}, s) & \text{otherwise} \end{cases},
\end{aligned}$$

where

$$\zeta_{\bar{\mathcal{A}}, \mathcal{A}}(\phi, u, a, \bar{a}, s) \equiv \begin{cases} \begin{cases} \phi_{\mathcal{A}}(u) & \phi_{\mathcal{A}}(u) \in \mathcal{Q}_{\mathcal{A}} \\ \bar{\rho}_{\bar{\mathcal{A}}}(\bar{a}, -l) & \phi_{\mathcal{A}}(u) = [r_{\mathcal{A}}, l] \in \mathbb{I}_{\mathcal{A}, r_{\mathcal{A}}} \\ \bar{a}, n_{\mathcal{A}} - |\mathcal{V}^-| + |\mathcal{V}^+|_u & \text{otherwise} \end{cases} & s = a \\ \begin{cases} \begin{cases} \bar{\rho}_{\mathcal{A}}(r_{\mathcal{A}}, -l) & [r_{\mathcal{A}}, -l] \in \mathcal{V}^- \\ \bar{a}, l - \varpi(\mathcal{V}^-, l) & l \in \langle n_{\mathcal{A}} \rangle \setminus \langle \omega(\mathcal{V}^-) \rangle \\ \bar{a}, l - |\mathcal{V}^-| + |\mathcal{V}^+| & \text{otherwise} \end{cases} & \phi_{\bar{\mathcal{A}}}(u) = [\bar{a}, l] \in \mathbb{I}_{\bar{\mathcal{A}}, \bar{a}} \\ \phi_{\bar{\mathcal{A}}}(u) & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

with

$$\begin{aligned}
\mathcal{U}^- &\subseteq \{\mathcal{X} \in \mathbb{I}_{\mathcal{A}, r_{\mathcal{A}}} \mid N_{\mathfrak{N}_{\bar{\mathcal{A}}}}^+(\mathcal{X}) \subseteq \mathbb{J}_{\mathcal{A}, a}\} & \mathcal{U}^+ &\subseteq \{\mathcal{X} \in \mathbb{I}_{\mathcal{A}, a} \mid N_{\mathfrak{N}_{\bar{\mathcal{A}}}}^+(\mathcal{X}) \not\subseteq \mathbb{J}_{\mathcal{A}, r_{\mathcal{A}}}\} \\
\mathcal{V}^- &\subseteq \{\mathcal{Y} \in \mathbb{J}_{\mathcal{A}, r_{\mathcal{A}}} \mid N_{\mathfrak{N}_{\bar{\mathcal{A}}}}^-(\mathcal{Y}) \subseteq \mathbb{I}_{\mathcal{A}, a}\} & \mathcal{V}^+ &\subseteq \{\mathcal{Y} \in \mathbb{J}_{\mathcal{A}, a} \mid N_{\mathfrak{N}_{\bar{\mathcal{A}}}}^-(\mathcal{Y}) \not\subseteq \mathbb{I}_{\mathcal{A}, r_{\mathcal{A}}}\}
\end{aligned}$$

and

$$\begin{aligned}\varpi(\mathcal{U}^\pm, k) &= |\{\mathcal{X} \in \mathcal{U}^\pm \mid \omega_{\mathcal{A}}(\mathcal{X}) \leq k\}| \\ \varpi(\mathcal{V}^\pm, k) &= |\{\mathcal{Y} \in \mathcal{V}^\pm \mid -\omega_{\mathcal{A}}(\mathcal{Y}) \leq k\}|,\end{aligned}$$

where \mathcal{A}' is defined by

$$\begin{aligned}\mathcal{Q}_{\mathcal{A}'} &= \pi_{\mathcal{A}}(\mathbb{S}_{\mathcal{A}} \setminus \{a\}) \cap \mathcal{Q}_{\mathcal{A}} \\ r_{\mathcal{A}'} &= r_{\mathcal{A}} \\ \mathbb{S}_{\mathcal{A}'} &= \mathbb{S}_{\mathcal{A}} \setminus \{a\} \\ \mu_{\mathcal{A}'}(s) &= \begin{cases} m_{\mathcal{A}} - |\mathcal{U}^-| + |\mathcal{U}^+| & s = r_{\mathcal{A}} \\ \mu_{\mathcal{A}}(s) & \text{otherwise} \end{cases} \\ \nu_{\mathcal{A}'}(s) &= \begin{cases} n_{\mathcal{A}} - |\mathcal{V}^-| + |\mathcal{V}^+| & s = r_{\mathcal{A}} \\ \nu_{\mathcal{A}}(s) & \text{otherwise} \end{cases} \\ \pi_{\mathcal{A}'}(s) &= \eta_{\mathcal{M}, \mathcal{A}}(\pi_{\mathcal{A}}(s), a) \\ \rho_{\mathcal{A}'}(s, -k) &= \begin{cases} \begin{cases} \bar{\rho}_{\mathcal{A}}(r_{\mathcal{A}}, -k - \varpi(\mathcal{V}^-, k)) & k \in \langle n_{\mathcal{A}} - |\mathcal{V}^-| \rangle \\ \bar{\rho}_{\mathcal{A}}(\mathcal{V}_{k-n_{\mathcal{A}}+|\mathcal{V}^-|}^+) & \text{otherwise} \end{cases} & s = r_{\mathcal{A}} \\ \eta_{\mathcal{M}, \mathcal{A}}(\rho_{\mathcal{A}}(s, -k), a) & \text{otherwise} \end{cases},\end{aligned}$$

where

$$\eta_{\mathcal{M}, \mathcal{A}}(u, a) \equiv \begin{cases} [r_{\mathcal{A}}, l - \varpi(\mathcal{U}^-, l)] & u = [r_{\mathcal{A}}, l] \in \mathbb{I}_{\mathcal{A}, r_{\mathcal{A}}} \\ [r_{\mathcal{A}}, m_{\mathcal{A}} - |\mathcal{U}^-| + |\mathcal{U}^+|_u] & u \in \mathbb{I}_{\mathcal{A}, a} \\ u & \text{otherwise} \end{cases},$$

then construct \mathcal{M}'''' from \mathcal{M}''' by duplicating the subnodes a and \bar{a} in $\bar{\mathcal{A}}'$ as a' and \bar{a}' in \mathcal{M}'''' and reassigning arguments and applicands so that the latter functionally replace the sources of the

former,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}''''} &= \mathcal{Q}_{\mathcal{M}'''} \cup \mathcal{Q}_{\bar{\mathcal{A}}'} \\
\tau_{\mathcal{M}''''} &= \tau_{\mathcal{M}'''} \\
\mathbb{S}_{\mathcal{M}''''} &= \mathbb{S}_{\mathcal{M}'''} \sqcup \{a', \bar{a}'\} \\
\mu_{\mathcal{M}''''}(s) &= \begin{cases} \mu_{\bar{\mathcal{A}}'}(a) & s = a' \\ \mu_{\bar{\mathcal{A}}'}(\bar{a}) & s = \bar{a}' \\ \mu_{\mathcal{M}'''}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}''''}(s) &= \begin{cases} \nu_{\bar{\mathcal{A}}'}(a) & s = a' \\ \nu_{\bar{\mathcal{A}}'}(\bar{a}) & s = \bar{a}' \\ \nu_{\mathcal{M}'''}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}''''}(s) &= \begin{cases} \vartheta_{\mathcal{M}''', \bar{\mathcal{A}}'}(\pi_{\bar{\mathcal{A}}'}(a), a, \bar{a}, \bar{a}', \bar{a}') & s = a' \\ \mathcal{A}' & s = \bar{a}' \\ \pi_{\mathcal{M}'''}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}''''}(s, -k) &= \begin{cases} \vartheta_{\mathcal{M}''', \bar{\mathcal{A}}'}(\rho_{\bar{\mathcal{A}}'}(a, -k), a, \bar{a}, \bar{a}', \bar{a}') & s = a' \\ \vartheta_{\mathcal{M}''', \bar{\mathcal{A}}'}(\rho_{\bar{\mathcal{A}}'}(\bar{a}, -k), \bar{a}, \bar{a}, a, a') & s = \bar{a}' \\ \begin{cases} \vartheta_{\mathcal{M}''', \bar{\mathcal{A}}'}(\rho_{\bar{\mathcal{A}}'}(a, -k), a, \bar{a}, \bar{a}', \bar{a}') & \bar{\rho}_{\bar{\mathcal{A}}'}(r_{\bar{\mathcal{A}}'}, -l) \in \mathbb{I}_{\bar{\mathcal{A}}', \bar{a}} \\ \vartheta_{\mathcal{M}''', \bar{\mathcal{A}}'}(\rho_{\bar{\mathcal{A}}'}(\bar{a}, -k), \bar{a}, \bar{a}, a, a') & \text{otherwise} \end{cases} & \rho_{\mathcal{M}'''}(s, -k) = [\bar{a}, l] \in \mathbb{I}_{\mathcal{M}''', \bar{a}} \\ \rho_{\mathcal{M}'''}(s, -k) & \text{otherwise} \end{cases}
\end{aligned}$$

where

$$\vartheta_{\mathcal{M}, \mathcal{A}}(u, a, \bar{a}, \bar{a}', \bar{a}') \equiv \begin{cases} u & u \in \mathcal{Q}_{\mathcal{A}} \\ \bar{\rho}_{\mathcal{M}}(\bar{a}, -l) & u = [r_{\mathcal{A}}, l] \in \mathbb{I}_{\mathcal{A}, r_{\mathcal{A}}} \\ \bar{a}' & u = \bar{a} \\ [\bar{a}', l] & u = [\bar{a}, l] \in \mathbb{I}_{\mathcal{A}, \bar{a}} \end{cases},$$

then if the subnode \bar{a} is not an applicand or argument in \mathcal{M}'''' , $\bar{a} \notin \pi_{\mathcal{M}''''}(\mathbb{U}_{\mathcal{M}''''}) \cup \rho_{\mathcal{M}''''}(\mathbb{Y}_{\mathcal{M}''''})$,

we remove it via $\mathcal{M}' = \tau_{\bar{a}}^{\text{SNL}}(\mathcal{M}''''')$, otherwise we construct $\mathcal{M}' = \nu_{\kappa, a}^{\text{RSP}} \nu_{\tau_{\mathcal{A}', \bar{a}}}^{\text{RSP}}(\mathcal{M}''''')$ where

$$\kappa = \begin{cases} r_{\mathcal{A}} & \pi_{\mathcal{A}}(a) \in \mathcal{Q}_{\mathcal{A}} \\ \pi_{\mathcal{M}'''''}(a') & \text{otherwise} \end{cases}$$

by reassigning the applicands of a and \bar{a} to be the root nodes of the submodel applicands of a' and \bar{a}' , respectively, or in the case of a' , possibly the non-submodel applicand of a' . We then rename a' as a .

Otherwise if $\mathcal{M} \prec_{\mathfrak{S}_{\mathcal{M}}} \bar{\mathcal{A}} \prec_{\mathfrak{S}_{\mathcal{M}}} \mathcal{A}$ and $\mathcal{T} = \mathcal{M}$, then $\mathcal{M}' = v_{a,\mathcal{T}}^{\text{RLS}}(v_{a,\bar{\mathcal{A}}}^{\text{RLS}}(\mathcal{M}))$ is constructed by first moving the subnode a outside \mathcal{A} into the submodel $\bar{\mathcal{A}}$ as in the previous case, then recursively applying the transformation to move a successively outward into \mathcal{M} .

Otherwise if $\mathcal{A} = \mathcal{M}$ and $\mathcal{M} \prec_{\mathfrak{S}_{\mathcal{M}}} \mathcal{T}$, we can define a construction similar to the above, moving the subnode a successively inward into \mathcal{T} . Finally, if $\mathcal{M} \prec_{\mathfrak{S}_{\mathcal{M}}} \mathcal{A}$ and $\mathcal{M} \prec_{\mathfrak{S}_{\mathcal{M}}} \mathcal{T}$, we can combine the outward procedure followed by the inward procedure.

This transformation effectively moves the subnode a somewhere in the internal models of \mathcal{M} and rearranges sources and sinks and reassigns arguments and applicands to preserve the functional interpretation of \mathcal{M} . In the visual representation, the subnode a and its contents are moved from inside the representation of \mathcal{A} to inside the representation of \mathcal{T} , and brackets and arrows throughout are added, removed, or relocated across the rectangle boundaries between the two locations. An additional subnode may also appear to preserve the submodel applicand of the original subnode containing a , when the latter is an applicand or argument of another node or sink.

Subnode Emersion (EMS) The model \mathcal{M} has a subset $\mathbb{W} \subseteq \mathfrak{S}_{\mathcal{M}}$ of its subnodes that is closed in the sense that there are no nodes of \mathcal{M} that precede one element of \mathbb{W} and succeed another element of \mathbb{W} in the node graph of \mathcal{M} , $\nexists_{s \in \mathbb{U}_{\mathcal{M}}} \exists_{t \in \mathbb{W}} \exists_{u \in \mathbb{W}} t \prec_{\mathfrak{R}_{\mathcal{M}}} s \prec_{\mathfrak{R}_{\mathcal{M}}} u$.

Then the model \mathcal{M}' in $\mathcal{N}' = v_{\mathcal{M},\mathbb{W}}^{\text{EMS}}(\mathcal{N})$ is constructed as follows: first form \mathcal{M}'' by creating a new subnode in \mathcal{M} with zero arity and coarity and null model applicand $\mathcal{S} = \mathcal{Z}$,

$$\begin{aligned} \mathcal{Q}_{\mathcal{M}''} &= \mathcal{Q}_{\mathcal{M}} \cup \{\mathcal{S}\} \\ \tau_{\mathcal{M}''} &= \tau_{\mathcal{M}} \\ \mathfrak{S}_{\mathcal{M}''} &= \mathfrak{S}_{\mathcal{M}} \sqcup \{t\} \\ \mu_{\mathcal{M}''}(s) &= \begin{cases} 0 & s = t \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \nu_{\mathcal{M}''}(s) &= \begin{cases} 0 & s = t \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \pi_{\mathcal{M}''}(s) &= \begin{cases} \mathcal{S} & s = t \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\ \rho_{\mathcal{M}''}(\mathcal{Y}) &= \rho_{\mathcal{M}}(\mathcal{Y}), \end{aligned}$$

then form a linear extension $\mathcal{W} \subseteq \mathbb{W}$ of \mathbb{W} with respect to the partial order $\prec_{\pi_{\mathcal{M}}}$ and form $\mathcal{M}' = \bigcirc_{t \in \mathcal{W}} v_{t, \mathcal{S}}^{\text{RLS}}(\mathcal{M}'')$ by relocating the subnodes \mathbb{W} into \mathcal{M}'' (our notation supposes that the modified submodel \mathcal{S} is relabeled \mathcal{S} after each application of RLS). This transformation effectively encapsulates the subnodes \mathbb{W} within the submodel applicand of a new subnode, rearranging arguments and applicands to leave the functional interpretation of \mathcal{M} unchanged. In the visual representation, the rectangles of the subnodes \mathbb{W} are enclosed in a new rectangle, and arrows crossing the boundary of the new rectangle are split by junctions across new brackets on its boundary. Note that the operator $v_{\mathcal{M}, \emptyset}^{\text{EMS}}$ simply creates a new “empty” subnode within the model \mathcal{M} .

Subnode Immersion (IMS) The model \mathcal{M} has a subnode $a \in \mathbb{S}_{\mathcal{M}}$ with a submodel applicand $\pi_{\mathcal{M}}(a) = \mathcal{S} \in \mathcal{Q}_{\mathcal{M}}$, and either a is neither the applicand of any subnode nor the argument of any sink in \mathcal{M} , $a \notin \pi_{\mathcal{M}}(\mathbb{S}_{\mathcal{M}}) \cup \rho_{\mathcal{M}}(\mathbb{Y}_{\mathcal{M}})$, or the applicand $\pi_{\mathcal{M}}(r_{\mathcal{M}}) \neq a$ and the arity $m_{\mathcal{S}} = 0$.

Then the model \mathcal{M}' in $\mathcal{N}' = v_a^{\text{IMS}}(\mathcal{N})$ is constructed as follows: first form \mathcal{M}'' by effectively expanding the root node $r_{\mathcal{S}}$ where applicable, so that $\mathcal{M}'' = \mathcal{M}$ if $r_{\mathcal{S}} \notin \pi_{\mathcal{S}}(\mathbb{U}_{\mathcal{S}}) \cup \rho_{\mathcal{S}}(\mathbb{Y}_{\mathcal{S}})$, otherwise $\pi_{\mathcal{M}}(a) = \mathcal{S}$ is replaced in \mathcal{M}'' by \mathcal{S}' , defined by

$$\begin{aligned}
\mathcal{Q}_{\mathcal{S}'} &= \mathcal{Q}_{\mathcal{S}} \cup \{\mathcal{S}\} \\
r_{\mathcal{S}'} &= r_{\mathcal{S}} \\
\mathbb{S}_{\mathcal{S}'} &= \mathbb{S}_{\mathcal{S}} \sqcup \{a'\} \\
\mu_{\mathcal{S}'}(s) &= \begin{cases} 0 & s = a' \\ \mu_{\mathcal{S}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{S}'}(s) &= \begin{cases} 0 & s = a' \\ \nu_{\mathcal{S}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{S}'}(s) &= \begin{cases} \mathcal{S} & s = a' \\ a' & \pi_{\mathcal{S}}(s) = r_{\mathcal{S}} \\ \pi_{\mathcal{S}}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{S}'}(\mathcal{Y}) &= \begin{cases} a' & \rho_{\mathcal{S}}(\mathcal{Y}) = r_{\mathcal{S}} \\ \rho_{\mathcal{S}}(\mathcal{Y}) & \text{otherwise} \end{cases};
\end{aligned}$$

then form $\mathcal{M}''' = v_a^{\text{SMA}}(\mathcal{M}'')$ by applying the submodel applicand of a at a ; finally, form \mathcal{M}' by eliminating the subnode a if applicable, so that $\mathcal{M}' = \mathcal{M}'''$ if $a \in \pi_{\mathcal{M}}(\mathbb{U}_{\mathcal{M}}) \cup \rho_{\mathcal{M}}(\mathbb{Y}_{\mathcal{M}}) \cup \chi_{\mathcal{M}}(\rho_{\mathcal{M}}(\mathbb{Y}_{\mathcal{M}}) \cap \mathbb{X}_{\mathcal{M}})$, otherwise $\mathcal{M}' = v_a^{\text{SNL}}(\mathcal{M}''')$.

Source Rearrangement (RRX) The model \mathcal{M} has an instance of an internal model \mathcal{A} with a node $a \in \mathbb{U}_{\mathcal{A}}$ with coarity $\nu_{\mathcal{A}}(a) = n$ that does not precede or succeed the root node $r_{\mathcal{M}}$ or one of its sources or sinks in the restricted node graph of \mathcal{M} , $N_{\mathfrak{M}_{\mathcal{M}}}^{\infty}(a) \cap (\mathbb{I}_{\mathcal{M}, r_{\mathcal{M}}} \cup \mathbb{J}_{\mathcal{M}, r_{\mathcal{M}}}) = \emptyset$.

Then the model \mathcal{M}' in $\mathcal{N}' = v_{a, i, j}^{\text{RRX}}(\mathcal{M})$ for $i, j \in \mathbb{N}$ is constructed by adding, removing, or reordering sources of a and the arguments or applicands equal to them, along with corresponding sources and sinks of successors or predecessors of a in the restricted node graph $\mathfrak{M}_{\mathcal{M}}$, so that the i th source of a becomes the j th source of a , where a new source is added when $i > n$, and a source is removed when $i \leq n < j$. In the latter case, we additionally require that if the applicand $\pi_{\mathcal{A}}(a) = \mathcal{Q} \in \mathcal{Q}_{\mathcal{A}}$ is a submodel, then the argument of the sink of $r_{\mathcal{Q}}$ corresponding to the removed source is undefined, $\rho_{\mathcal{Q}}(r_{\mathcal{Q}}, -i) = \emptyset$; and if the node $a = r_{\mathcal{A}}$, then the removed source is not the applicand or argument of any node or sink in \mathcal{A} , $[a, i] \notin \pi_{\mathcal{A}}(\mathbb{U}_{\mathcal{A}}) \cup \rho_{\mathcal{A}}(\mathbb{Y}_{\mathcal{A}})$.

We construct the model \mathcal{M}' as follows: first form the model $\mathcal{M}'' = v_{a, i, j}^{\text{RSX}}(\mathcal{M})$ by reassigning the sources of a ; then

1. if $a = r_{\mathcal{A}}$ is a root node, then: if \mathcal{A} has an immediate predecessor $\bar{a} \in N_{\mathfrak{S}_{\mathcal{M}}}^{-}(\mathcal{A})$ in the submodel graph of \mathcal{M} , form $\mathcal{M}''' = v_{\bar{a}, i, j}^{\text{RRY}+}(v_{\bar{a}, i, j}^{\text{RRY}-}(v_{\bar{a}, i, j}^{\text{RSY}}(\mathcal{M}'')))$ by rearranging the corresponding sinks of \bar{a} and corresponding sources or sinks of its successors and predecessors in the applicand graph $\mathfrak{A}_{\mathcal{M}}$, otherwise let $\mathcal{M}''' = \mathcal{M}''$; then form $\mathcal{M}' = v_{a, i, j}^{\text{RRY}+}(v_{a, i, j}^{\text{RRY}-}(\mathcal{M}'''))$ by rearranging the corresponding sources or sinks of the successors and predecessors of a in $\mathfrak{A}_{\mathcal{M}}$; otherwise
2. if $a \in \mathbb{S}_{\mathcal{A}}$ is a subnode, then: if $\pi_{\mathcal{A}}(a) = \mathcal{Q} \in \mathcal{Q}_{\mathcal{A}}$, form $\mathcal{M}''' = v_{r_{\mathcal{Q}}, i, j}^{\text{RRY}+}(v_{r_{\mathcal{Q}}, i, j}^{\text{RRY}-}(v_{r_{\mathcal{Q}}, i, j}^{\text{RSY}}(\mathcal{M}'')))$ by rearranging the corresponding sinks of the root node $r_{\mathcal{Q}}$ and its successors and predecessors in the applicand graph $\mathfrak{A}_{\mathcal{M}}$, otherwise let $\mathcal{M}''' = \mathcal{M}''$; then form $\mathcal{M}' = v_{a, i, j}^{\text{RRX}+}(v_{a, i, j}^{\text{RRX}-}(\mathcal{M}'''))$ by rearranging the corresponding sources or sinks of the successors and predecessors of a in $\mathfrak{A}_{\mathcal{M}}$,

where

- the model $v_{a, i, j}^{\text{RRX}+}(\mathcal{M}) \equiv \bigcirc_{s \in \mathcal{R}_{\mathcal{M}}} \bigcirc_{s \in \mathbb{U}_{\mathcal{S}} \cap N_{\mathfrak{A}_{\mathcal{M}}}^{+}(a)} v_{s, i^{+}, j^{+}}^{\text{RRX}+} \circ v_{s, i^{+}, j^{+}}^{\text{XYZ}}(\mathcal{M})$, with $\text{XYZ} = \begin{cases} \text{RRY} & s = r_{\mathcal{S}} \\ \text{RRX} & s \in \mathbb{S}_{\mathcal{S}} \end{cases}$ and $k^{+} = k - \varrho_{\mathcal{A}}(a, -k)$ for $k \in \{i, j\}$, is defined recursively to rearrange the corresponding sources of subnodes and corresponding sinks of root nodes among successors of a in $\mathfrak{A}_{\mathcal{M}}$
- the model $v_{a, i, j}^{\text{RRX}-}(\mathcal{M}) \equiv \bigcirc_{s \in \mathcal{R}_{\mathcal{M}}} \bigcirc_{s \in \mathbb{U}_{\mathcal{S}} \cap N_{\mathfrak{A}_{\mathcal{M}}}^{-}(a)} v_{s, i^{-}, j^{-}}^{\text{RRX}-} \circ v_{s, i^{-}, j^{-}}^{\text{XYZ}}(\mathcal{M})$, with $\text{XYZ} = \begin{cases} \text{RRY} & s = r_{\mathcal{S}} \\ \text{RRX} & s \in \mathbb{S}_{\mathcal{S}} \end{cases}$ and k^{-} such that $\varrho_{\mathcal{S}}(s, -k^{-}) = k$ for $k \in \{i, j\}$, is defined similarly for the predecessors of a in $\mathfrak{A}_{\mathcal{M}}$

where in the above, we additionally require that the preconditions on RRX are satisfied for each recursive application of RRX+ and RRX-, and similarly for RRY.

This transformation effectively inserts, deletes, or permutes one source of a without changing the external connections to existing sources of a that remain, and propagates this rearrangement to the sources and sinks of related nodes of internal models of \mathcal{M} to preserve the functional interpretation of \mathcal{M} . In the visual representation, one bracket on the right side (left side) of the rectangle of the subnode (root node) a is added, removed, or moved to a different position on the side of the rectangle along with the tail of any arrows out of it, and similar changes are made to nodes that precede or succeed a in the restricted node graph $\underline{\mathfrak{N}}_{\mathcal{M}}$.

Sink Rearrangement (RRY) The model \mathcal{M} has an instance of an internal model \mathcal{A} with a node $a \in \mathbb{U}_{\mathcal{A}}$ with arity $\mu_{\mathcal{A}}(a) = m$ that does not precede or succeed the root node $r_{\mathcal{M}}$ or one of its sources or sinks in the restricted node graph of \mathcal{M} , $N_{\underline{\mathfrak{N}}_{\mathcal{M}}}^{\infty}(a) \cap (\mathbb{I}_{\mathcal{M}, r_{\mathcal{M}}} \cup \mathbb{J}_{\mathcal{M}, r_{\mathcal{M}}}) = \emptyset$.

Then the model \mathcal{M}' in $\mathcal{N}' = v_{a, i, j}^{\text{RRY}}(\mathcal{N})$ for $i, j \in \mathbb{N}$ is constructed by adding, removing, or reordering sinks of a and their arguments, along with corresponding sources and sinks of successors or predecessors of a in the restricted node graph $\underline{\mathfrak{N}}_{\mathcal{M}}$, so that the i th sink of a becomes the j th sink of a , where a new sink is added when $i > m$, and a sink is removed when $i \leq m < j$. In the latter case, we additionally require that if the applicand $\pi_{\mathcal{A}}(a) = \mathcal{Q} \in \mathcal{Q}_{\mathcal{A}}$ is a submodel, then the source of $r_{\mathcal{Q}}$ corresponding to the removed sink is not the applicand or argument of any node or sink in \mathcal{Q} , $[r_{\mathcal{Q}}, i] \notin \pi_{\mathcal{Q}}(\mathbb{U}_{\mathcal{Q}}) \cup \rho_{\mathcal{Q}}(\mathbb{Y}_{\mathcal{Q}})$; and if the node $a = r_{\mathcal{A}}$, then the argument of the removed sink is undefined, $\rho_{\mathcal{A}}(a, -i) = \emptyset$.

We construct the model \mathcal{M}' similarly to the analogous construction for RRX above, generally by substituting RRY for RRX and vice versa. The additional preconditions discussed above apply in an analogous fashion. The interpretation is analogous, substituting sinks for sources, and the visual representation is analogous, substituting the left side of a rectangle for the right side and vice versa.

Argument/Applicand Rearrangement (RRP) The model \mathcal{M} has instances of an internal model $\mathcal{A} \in \mathcal{R}_{\mathcal{M}}$ with a node $a \in \mathbb{U}_{\mathcal{M}}$ and of an internal model $\mathcal{T} \in \mathcal{R}_{\mathcal{M}}$, and either

1. a node or source $\varkappa \in \mathbb{I}_{\mathcal{M}, a}$ or
2. a node or sink $\varsigma \in \mathbb{J}_{\mathcal{M}, a}$.

Then the model \mathcal{M}' in $\mathcal{N}' = v_{\varkappa, \mathcal{T}}^{\text{RRP}^+}(\mathcal{N})$ or $\mathcal{N}' = v_{\varsigma, \mathcal{T}}^{\text{RRP}^-}(\mathcal{N})$ is constructed by

1. forming the model \mathcal{M}'' by undefining any arguments and applicands of nodes or sinks

$$\mathcal{K} \subseteq \{u \in \mathbb{U}_{\mathcal{M}} \mid \pi_{\mathcal{M}}(u) = \varkappa\} \sqcup \{\mathcal{Y} \in \mathbb{Y}_{\mathcal{M}} \mid \rho_{\mathcal{M}}(\mathcal{Y}) = \varkappa\}$$

equal to \varkappa and forming a new subnode t with a submodel applicand \mathcal{S} functionally restoring

these arguments and applicands,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}''} &= \mathcal{Q}_{\mathcal{M}} \cup \{\mathcal{S}\} \\
r_{\mathcal{M}''} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}''} &= \mathbb{S}_{\mathcal{M}} \sqcup \{t\} \\
\mu_{\mathcal{M}''}(s) &= \begin{cases} 1 & s = t \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}''}(s) &= \begin{cases} |\mathcal{K}| & s = t \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}''}(s) &= \begin{cases} \mathcal{S} & s = t \\ [t, |\mathcal{K}|_s] & \pi_{\mathcal{M}}(s) = \varkappa \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}''}(\mathcal{Y}) &= \begin{cases} \varkappa & \mathcal{Y} = [t, -1] \\ [t, |\mathcal{K}|_{\mathcal{Y}}] & \rho_{\mathcal{M}}(\mathcal{Y}) = \varkappa \\ \rho_{\mathcal{M}}(\mathcal{Y}) & \text{otherwise} \end{cases}
\end{aligned}$$

where

$$\begin{aligned}
\mathcal{Q}_{\mathcal{S}} &= \emptyset \\
r_{\mathcal{S}} &= r_{\mathcal{S}} \\
\mathbb{S}_{\mathcal{S}} &= \emptyset \\
\mu_{\mathcal{S}}(r_{\mathcal{S}}) &= |\mathcal{K}| \\
\nu_{\mathcal{S}}(r_{\mathcal{S}}) &= 1 \\
\pi_{\mathcal{S}}(r_{\mathcal{S}}) &= \emptyset \\
\rho_{\mathcal{S}}(r_{\mathcal{S}}, k) &= [r_{\mathcal{S}}, 1],
\end{aligned}$$

or

2. forming the model \mathcal{M}'' by undefining the argument or applicand of ς and forming a new

subnode t with a submodel applicand \mathcal{S} functionally restoring this argument or applicand,

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}''} &= \mathcal{Q}_{\mathcal{M}} \cup \{\mathcal{S}\} \\
r_{\mathcal{M}''} &= r_{\mathcal{M}} \\
\mathbb{S}_{\mathcal{M}''} &= \mathbb{S}_{\mathcal{M}} \sqcup \{t\} \\
\mu_{\mathcal{M}''}(s) &= \begin{cases} 1 & s = t \\ \mu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}''}(s) &= \begin{cases} 1 & s = t \\ \nu_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}''}(s) &= \begin{cases} \mathcal{S} & s = t \\ [t, 1] & s = \varsigma \\ \pi_{\mathcal{M}}(s) & \text{otherwise} \end{cases} \\
\rho_{\mathcal{M}''}(\mathcal{Y}) &= \begin{cases} \begin{cases} \pi_{\mathcal{M}}(\varsigma) & \varsigma \in \mathbb{U}_{\mathcal{M}} \\ \rho_{\mathcal{M}}(\varsigma) & \varsigma \in \mathbb{Y}_{\mathcal{M}} \end{cases} & \mathcal{Y} = [t, -1] \\ [t, 1] & \mathcal{Y} = \varsigma \\ \rho_{\mathcal{M}}(\mathcal{Y}) & \text{otherwise} \end{cases}
\end{aligned}$$

where

$$\begin{aligned}
\mathcal{Q}_{\mathcal{S}} &= \emptyset \\
r_{\mathcal{S}} &= r_{\mathcal{S}} \\
\mathbb{S}_{\mathcal{S}} &= \emptyset \\
\mu_{\mathcal{S}}(r_{\mathcal{S}}) &= 1 \\
\nu_{\mathcal{S}}(r_{\mathcal{S}}) &= 1 \\
\pi_{\mathcal{S}}(r_{\mathcal{S}}) &= \emptyset \\
\rho_{\mathcal{S}}(r_{\mathcal{S}}, -1) &= [r_{\mathcal{S}}, 1],
\end{aligned}$$

then relocating t into \mathcal{T} and immersing it in \mathcal{T} , $\mathcal{M}' = v_t^{\text{IMS}}(v_{t, \mathcal{T}}^{\text{RLS}}(\mathcal{M}''))$. This transformation effectively moves a portion of the path of the connection(s) represented by the argument(s) or applicand(s) of ς or equal to \varkappa into the internal model \mathcal{T} . In the visual representation, the midpoint(s) of the corresponding arrow(s) are moved into the representation of \mathcal{T} , with the connecting path(s) preserved by arrows and junctions of arrows across left and right sides of rectangles.

One may verify by methods similar to those in §3.3.2 that the syntactic transformations defined above do not alter the normal form of models to which they are applied.

4.3 Partial Inversion Transformation

The partial inversion transformation interchanges the roles of root node sources and sinks (functionally, inputs and outputs) in a model, changing the model's normal form but preserving certain mathematical relationships between its elements. Partial inversion captures the operations that would normally be performed in engineering design activities through offline equation solving (manually or by computer algebra) or numerical iteration, but can often provide more useful results in practice (§5.3).

In its most general form, a partial inversion of a model \mathcal{N} with $\pi_{\mathcal{N}}(r_{\mathcal{N}}) = r_{\mathcal{N}}$ by a permutation $\sigma : \langle\langle -m_{\mathcal{N}} \rangle\rangle \sqcup \langle\langle n_{\mathcal{N}} \rangle\rangle \longleftrightarrow \langle\langle -m_{\mathcal{N}} \rangle\rangle \sqcup \langle\langle n_{\mathcal{N}} \rangle\rangle$ is a model \mathcal{N}' such that the model \mathcal{M} defined by

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}} &= \{\mathcal{N}, \mathcal{N}'\} \\
r_{\mathcal{M}} &= r_{\mathcal{M}} \\
\mathcal{S}_{\mathcal{M}} &= \{a, a'\} \\
\mu_{\mathcal{M}}(s) &= \begin{cases} n_{\mathcal{N}} & s = r_{\mathcal{M}} \\ m_{\mathcal{N}} & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}}(s) &= \begin{cases} m_{\mathcal{N}} & s = r_{\mathcal{M}} \\ n_{\mathcal{N}} & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}}(s) &= \begin{cases} \emptyset & s = r_{\mathcal{M}} \\ \mathcal{N} & s = a \\ \mathcal{N}' & s = a' \end{cases} \\
\rho_{\mathcal{M}}(s, -k) &= \begin{cases} [a', k] & s = r_{\mathcal{M}} \\ [r_{\mathcal{M}}, k] & s = a \\ \begin{cases} [r_{\mathcal{M}}, -\sigma(-k)] & \sigma(-k) \in \langle -m_{\mathcal{N}} \rangle \\ [a, \sigma(-k)] & \sigma(-k) \in \langle n_{\mathcal{N}} \rangle \end{cases} & s = a' \end{cases}
\end{aligned}$$

has the same normal form as the model \mathcal{M}' defined by

$$\begin{aligned}
\mathcal{Q}_{\mathcal{M}'} &= \{\mathcal{N}\} \\
r_{\mathcal{M}'} &= r_{\mathcal{M}'} \\
\mathcal{S}_{\mathcal{M}'} &= \{a\} \\
\mu_{\mathcal{M}'}(s) &= \begin{cases} n_{\mathcal{N}} & s = r_{\mathcal{M}'} \\ m_{\mathcal{N}} & \text{otherwise} \end{cases} \\
\nu_{\mathcal{M}'}(s) &= \begin{cases} m_{\mathcal{N}} & s = r_{\mathcal{M}'} \\ n_{\mathcal{N}} & \text{otherwise} \end{cases} \\
\pi_{\mathcal{M}'}(s) &= \begin{cases} \emptyset & s = r_{\mathcal{M}'} \\ \mathcal{N} & s = a \end{cases} \\
\rho_{\mathcal{M}'}(s, -k) &= \begin{cases} \begin{cases} [r_{\mathcal{M}'}, -\sigma(k)] & \sigma(k) \in \langle -m_{\mathcal{N}} \rangle \\ [a, \sigma(k)] & \sigma(k) \in \langle n_{\mathcal{N}} \rangle \end{cases} & s = r_{\mathcal{M}'} \\ [r_{\mathcal{M}'}, k] & s = a \end{cases}
\end{aligned}$$

In the functional interpretation, this roughly corresponds to the statement that, under certain conditions, when \mathcal{N} takes inputs $\{x_i\}_{i \in \langle -m_{\mathcal{N}} \rangle}$ and gives outputs $\{x_i\}_{i \in \langle n_{\mathcal{N}} \rangle}$, then \mathcal{N}' gives outputs $\{x_i\}_{i \in \sigma(\langle n_{\mathcal{N}} \rangle)}$ when given inputs $\{x_i\}_{i \in \sigma(\langle m_{\mathcal{N}} \rangle)}$. There is no algorithm to compute a partial inversion of a model \mathcal{N} in the general case; we describe an algorithm for computing partial inversions in some special cases useful in engineering design.

We begin with the case with primitives $\mathbb{P} = \mathbb{C} \sqcup \{+, \cdot\}$, with the natural auxiliary normalization rules corresponding to the n -ary addition and multiplication operators, with the restriction that \mathcal{N} is mathematically well-formed in its functional interpretation, $\mathbb{C} \cap \mathbb{H}_{\mathcal{N}} = \emptyset$ and $\forall_{S \in \mathcal{R}_{\mathcal{N}}} \forall_{s \in \mathbb{U}_S} s \in \mathbb{H}_{\mathcal{N}} \Rightarrow (\mathbb{I}_{S,s} \sqcup \mathbb{J}_{S,s}) \cap \mathbb{H}_{\mathcal{N}} = \emptyset$, where $\mathbb{H}_{\mathcal{N}} \equiv \bigsqcup_{S \in \mathcal{R}_{\mathcal{N}}} \{\varkappa \in \mathbb{U}_S \sqcup \mathbb{X}_S \sqcup \mathbb{Y}_S \sqcup \mathbb{P} \mid \exists_{p \in \{+, \cdot\}} p \prec_{\underline{\mathfrak{M}}_{\mathcal{M}}} \varkappa \vee \exists_{s \in \bigsqcup_{\mathcal{R} \in \mathcal{R}_{\mathcal{N}}} \mathbb{U}_{\mathcal{R}}} \varkappa \prec s\}$ consists of the elements of the restricted node graph $\underline{\mathfrak{M}}_{\mathcal{M}}$ associated with polynomial operators in the functional interpretation. This restriction yields models whose functional interpretations are polynomial functions with complex coefficients.

We can compute partial inversions of \mathcal{N} by permutations σ that rearrange only those sources and sinks that do not correspond to operators, $\forall_{[r_{\mathcal{N}}, -k] \in (\mathbb{I}_{\mathcal{N}, r_{\mathcal{N}}} \sqcup \mathbb{J}_{\mathcal{N}, r_{\mathcal{N}}}) \cap \mathbb{H}_{\mathcal{N}}} \sigma(k) = k$, subject to certain conditions discussed below. In the functional interpretation, this amounts to solving the corresponding polynomial systems for certain variables in terms of others. For our purposes, we also assume that the polynomial systems are initialized to certain values of the variables. Such partial inversions of \mathcal{N} are computed by the following approach:

1. Construct a bigraph \mathfrak{B} on the set $(\mathbb{X}_{\mathcal{N}} \sqcup \mathbb{Y}_{\mathcal{N}}) \cap \mathbb{H}_{\mathcal{N}}$ of sources and sinks not corresponding to

operators and the set $\mathbb{S}_{\mathcal{N}}$ of subnodes, in which the edges join subnodes to their sources and sinks. This bigraph is continually maintained through incremental updates as the model \mathcal{N} is modified by transformations.

2. Compute the block triangular decomposition of \mathfrak{B} (§2.2.4). If it has horizontal components, the partial inversion does not exist. If it has vertical components, discard an arbitrary subset of rows in the corresponding binary matrix to make these components square, then check consistency at the end (step 4). This block triangular decomposition is also incrementally maintained at each step of updating the bigraph \mathfrak{B} , by iterating the augmentation step in the maximum matching subroutine and recomputing those strongly connected components in the corresponding digraph whose boundary or interior has been changed.
3. For each component of the block triangular decomposition:
 - (a) If the component consists of a single subnode with a submodel applicand, recursively compute the partial inversion of the submodel corresponding to the desired permutation and move to the next component; otherwise, compute the corresponding polynomial system of the subnode.
 - (b) Compute the Gröbner basis of the polynomial system corresponding to the component and extract the roots in successive variables in terms of prior variables (§2.1.2), using a compact representation of univariate roots (§5.1). Select the root functions corresponding to the current initialization of the variables.
 - (c) The new subnodes corresponding to this component correspond to the selected root functions, with links corresponding to each variable in the polynomial system.
4. Combine the new subnodes generated from each component and add links between them as indicated in \mathfrak{B} to construct the partial inversion of \mathcal{N} . Check consistency of any discarded rows of \mathfrak{B} : if they are consistent, the partial inversion exists, otherwise it does not.

The above approach completely solves the partial inversion transformation in the polynomial case. For the more general case where models correspond to arbitrary invertible functions, the approach is unchanged except for step 3(b). Here the Gröbner basis approach can be applied for some generalizations of polynomial systems, such as nonlinear integer programming [7]. In more general cases, with certain special functions, one can apply cylindrical algebraic decomposition, though this can quickly become computationally intractable for large models [2, 14]. When models correspond to Lipschitz-continuous functions, this approach can be augmented with seminumerical methods to obtain primitive subroutines substituting for some or all partial inversions.

We explain this approach in more detail by example, via the case study in §5.2. Some of the constituent algorithms and their efficient implementation are the subject of much active research:

§6.3 summarizes some recent developments. However, the overall procedure described here has several generic benefits in the context of engineering design, as discussed later on.

The model transformations presented in this chapter provide the practical counterpart to the structured representation of MDPL models introduced in Chapter 3. It is the constrained structure of MDPL that gives the language its expressive power to create and manipulate large programmatic constructs. By itself, it would pose serious challenges to a programmer wishing to change the relationship between distant elements of a model. The model transformations automatically navigate and preserve the model structure in order to make such changes possible with a single operation.

Chapter 5

Theoretical & Experimental Results

We now discuss the practical use of MDPL to solve real-world engineering problems, and its advantages as the basis of a general-purpose programming language. In §5.1 we describe a prototype environment implementing the core formalism of MDPL, with some additional features to make it more readily applicable to engineering problems. This prototype environment has been used in an experimental case study comparing its performance with that of a traditional spreadsheet environment in helping real users perform a nontrivial conceptual design analysis task, reported in §5.2. This case study brings out some of the practical advantages of MDPL as an engineering tool, which we discuss further in §5.3 along with broader consideration of their theoretical significance. The case study also highlights some opportunities for further research and development, which we discuss along with past and current related work in Chapter 6.

5.1 Prototype Development Environment

The prototype MDPL environment used in the experiment of §5.2 provides a crude graphical user interface (GUI) to a single MDPL model \mathcal{N} (termed the “current” model), including the ability to specify and execute the model transformations in Chapter 4 on \mathcal{N} , and to evaluate \mathcal{N} to its normal form as described in Chapter 3. It also provides certain primitives with predefined normalization rules that make the language more readily applicable to engineering problems. Finally, it provides some features for translating traditional mathematical syntax into MDPL models; this helps to overcome some limitations of the immature user interface in creating large models from scratch. This prototype environment is implemented in *Mathematica 6*, making extensive use of its extensive symbolic language and user interface capabilities.

The GUI represents the current model using a version the visual representation described in §3.2, limited by the effort invested in software development to date. At any one time, the viewing area shows either the entire model, or one of its internal models, termed the “focused” model \mathcal{F} . All

subnodes of \mathcal{F} are identified only by labels that may be assigned to them: no submodel applicand is visible until the user focuses on it. For any subnode in \mathcal{F} with a submodel applicand, the user may focus on that submodel by double-clicking the pointer on the subnode. For an instance of an internal model of the current model that is the applicand of a subnode of a “parent” model, the user may focus on the parent model by holding down the SHIFT key and double-clicking the pointer anywhere in the window.

To apply one of the model transformations in Chapter 4, the user may select certain elements of \mathcal{F} (nodes, sources, or sinks) by clicking on their visual representations with the pointer. The user may select multiple elements simultaneously by holding down the CTRL key while clicking. If an element is selected in \mathcal{F} , all selected elements in any other internal model of the current model are deselected. The user may deselect all elements by clicking the pointer in an area outside the representation of any element of \mathcal{F} , termed clicking “in” \mathcal{F} . Once certain elements are selected in a model \mathcal{M} , the user may perform certain keyboard or pointer actions to apply the desired transformation. The transformations performed by various user interface actions are summarized in Table 5.1, where $\mathbb{U}^* \subseteq \mathbb{U}_{\mathcal{M}}$ is the set of selected nodes, $\mathbb{X}^* \subseteq \mathbb{X}_{\mathcal{M}}$ is the set of selected sources, and $\mathbb{Y}^* \subseteq \mathbb{Y}_{\mathcal{M}}$ is the set of selected sinks.

Some user interface actions apply multiple transformations in a compound fashion, reducing the number of operations required to complete some common tasks. For instance, applying the UP ARROW or DOWN ARROW keys to multiple sources and sinks moves all selected sources and sinks up or down in the source/sink orderings of their corresponding subnodes (where applicable). Similarly, the DELETE and ENTER keys remove all selected subnodes, sources, and sinks, or add new sources and sinks below all selected sources and sinks, respectively. Combining the CTRL key with the previous keys applies the syntactic transformations RRX and RRY in place of the semantic transformations RSX and RSY. Right-clicking on a source or node generates paths of arrows from it to all selected nodes and sinks. Right-clicking in \mathcal{M} encapsulates the selected subnodes via EMS (when applicable), while right-clicking in $\mathcal{F} \neq \mathcal{M}$ relocates the selected subnodes to \mathcal{F} via RLS. The remaining transformations can be applied similarly to all applicable selected elements of \mathcal{M} .

Where possible, the set of selected elements of each type persists after a transformation is applied, unless specifically cleared by the user. This allows multiple related transformations to be applied quickly to the same element or group of elements. The last column of Table 5.1 indicates when certain types of elements must be cleared from the selection by the application of a transformation, in order to maintain the restriction of the selection to elements of a single model. It may prove useful in future MDPL interfaces to relax this restriction in some cases. Certainly there is much room for improvement upon this interface design, which was limited by the time available for software development. Some ideas from participants in the case study of §5.2 are described later on.

The prototype environment used in the case study of §5.2 provides some additional built-in

Table 5.1: Model transformations applied by user interface actions in the prototype environment

UI action	Change to model \mathcal{N}	Selection subsets cleared
DELETE	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{U}^* \cap \mathbb{S}_{\mathcal{M}}} v_s^{\text{DLS}} \circ \bigcirc_{s \in \mathbb{S}_{\mathcal{M}}} \left(\bigcirc_{k \in \langle \nu_{\mathcal{M}}(s) \rangle [s, k] \in \mathbb{X}^*} v_{s, k, \nu_{\mathcal{M}}(s)+1}^{\text{RSX}} \right. \\ \left. \bigcirc_{k \in \langle \mu_{\mathcal{M}}(s) \rangle [s, -k] \in \mathbb{Y}^*} v_{s, k, \mu_{\mathcal{M}}(s)+1}^{\text{RSY}} \right)$	$\mathbb{U}^*, \mathbb{X}^*, \mathbb{Y}^*$
UP ARROW	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{S}_{\mathcal{M}}} \left(\bigcirc_{k \in \langle \nu_{\mathcal{M}}(s) \rangle k > 1 \wedge [s, k] \in \mathbb{X}^*} v_{s, k, k-1}^{\text{RSX}} \right. \\ \left. \bigcirc_{k \in \langle \mu_{\mathcal{M}}(s) \rangle k > 1 \wedge [s, -k] \in \mathbb{Y}^*} v_{s, k, k-1}^{\text{RSY}} \right) (\mathcal{N})$	–
DOWN ARROW	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{S}_{\mathcal{M}}} \left(\bigcirc_{k - \nu_{\mathcal{M}}(s) - 1 \in \langle -\nu_{\mathcal{M}}(s) \rangle k < \nu_{\mathcal{M}}(s) \wedge [s, k] \in \mathbb{X}^*} v_{s, k, k+1}^{\text{RSX}} \right. \\ \left. \bigcirc_{k - \mu_{\mathcal{M}}(s) - 1 \in \langle -\mu_{\mathcal{M}}(s) \rangle k < \mu_{\mathcal{M}}(s) \wedge [s, -k] \in \mathbb{Y}^*} v_{s, k, k+1}^{\text{RSY}} \right) (\mathcal{N})$	–
ENTER	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{S}_{\mathcal{M}}} \left(\bigcirc_{k - \nu_{\mathcal{M}}(s) - 1 \in \langle -\nu_{\mathcal{M}}(s) \rangle k < \nu_{\mathcal{M}}(s) \wedge [s, k] \in \mathbb{X}^*} v_{s, 2 \cdot \nu_{\mathcal{M}}(s), k+1}^{\text{RSX}} \right. \\ \left. \bigcirc_{k - \mu_{\mathcal{M}}(s) - 1 \in \langle -\mu_{\mathcal{M}}(s) \rangle k < \mu_{\mathcal{M}}(s) \wedge [s, -k] \in \mathbb{Y}^*} v_{s, 2 \cdot \mu_{\mathcal{M}}(s), k+1}^{\text{RSY}} \right) (\mathcal{N})$	–
CTRL+DELETE	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{U}^* \cap \mathbb{S}_{\mathcal{M}}} v_s^{\text{DLS}} \circ \bigcirc_{s \in \mathbb{S}_{\mathcal{M}}} \left(\bigcirc_{k \in \langle \nu_{\mathcal{M}}(s) \rangle [s, k] \in \mathbb{X}^*} v_{s, k, \nu_{\mathcal{M}}(s)+1}^{\text{RRX}} \right. \\ \left. \bigcirc_{k \in \langle \mu_{\mathcal{M}}(s) \rangle [s, -k] \in \mathbb{Y}^*} v_{s, k, \mu_{\mathcal{M}}(s)+1}^{\text{RRY}} \right) (\mathcal{N})$	$\mathbb{U}^*, \mathbb{X}^*, \mathbb{Y}^*$
CTRL+UP ARROW	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{S}_{\mathcal{M}}} \left(\bigcirc_{k \in \langle \nu_{\mathcal{M}}(s) \rangle k > 1 \wedge [s, k] \in \mathbb{X}^*} v_{s, k, k-1}^{\text{RRX}} \right. \\ \left. \bigcirc_{k \in \langle \mu_{\mathcal{M}}(s) \rangle k > 1 \wedge [s, -k] \in \mathbb{Y}^*} v_{s, k, k-1}^{\text{RRY}} \right) (\mathcal{N})$	–
CTRL+DOWN ARROW	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{S}_{\mathcal{M}}} \left(\bigcirc_{k - \nu_{\mathcal{M}}(s) - 1 \in \langle -\nu_{\mathcal{M}}(s) \rangle k < \nu_{\mathcal{M}}(s) \wedge [s, k] \in \mathbb{X}^*} v_{s, k, k+1}^{\text{RRX}} \right. \\ \left. \bigcirc_{k - \mu_{\mathcal{M}}(s) - 1 \in \langle -\mu_{\mathcal{M}}(s) \rangle k < \mu_{\mathcal{M}}(s) \wedge [s, -k] \in \mathbb{Y}^*} v_{s, k, k+1}^{\text{RRY}} \right) (\mathcal{N})$	–
CTRL+ENTER	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{S}_{\mathcal{M}}} \left(\bigcirc_{k - \nu_{\mathcal{M}}(s) - 1 \in \langle -\nu_{\mathcal{M}}(s) \rangle k < \nu_{\mathcal{M}}(s) \wedge [s, k] \in \mathbb{X}^*} v_{s, 2 \cdot \nu_{\mathcal{M}}(s), k+1}^{\text{RRX}} \right. \\ \left. \bigcirc_{k - \mu_{\mathcal{M}}(s) - 1 \in \langle -\mu_{\mathcal{M}}(s) \rangle k < \mu_{\mathcal{M}}(s) \wedge [s, -k] \in \mathbb{Y}^*} v_{s, 2 \cdot \mu_{\mathcal{M}}(s), k+1}^{\text{RRY}} \right) (\mathcal{N})$	–
right-click on $\varkappa \in \mathbb{U}_{\mathcal{F}} \sqcup \mathbb{X}_{\mathcal{F}}$	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{U}^*} v_{\varkappa, s}^{\text{RSP}} \circ \bigcirc_{\gamma \in \mathbb{Y}^*} v_{\varkappa, \gamma}^{\text{RSR}} (\mathcal{N})$	–
right-click in $\mathcal{F} \neq \mathcal{M}$	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{U}^*} v_{s, \mathcal{F}}^{\text{RLS}} (\mathcal{N})$	$\mathbb{X}^*, \mathbb{Y}^*$
right-click in $\mathcal{F} = \mathcal{M}$	$\mathcal{N} \leftarrow v_{\mathcal{M}, \mathbb{U}^* \cap \mathbb{S}_{\mathcal{M}}}^{\text{EMS}} (\mathcal{N})$	$\mathbb{U}^*, \mathbb{X}^*, \mathbb{Y}^*$
SHIFT+right-click in $\mathcal{F} = \mathcal{M}$	$\mathcal{N} \leftarrow \bigcirc_{s \in \mathbb{U}^* \cap \mathbb{S}_{\mathcal{M}}} v_s^{\text{IMS}} (\mathcal{N})$	\mathbb{U}^*
CTRL+left-click in \mathcal{F}	$\mathcal{N} \leftarrow \bigcirc_{\zeta \in \mathbb{U}^* \cup \mathbb{Y}^*} v_{\zeta, \mathcal{F}}^{\text{RRP}^-} (\mathcal{N})$	–
CTRL+right-click in \mathcal{F}	$\mathcal{N} \leftarrow \bigcirc_{\zeta \in \mathbb{U}^* \cup \mathbb{X}^*} v_{\zeta, \mathcal{F}}^{\text{RRP}^+} (\mathcal{N})$	–

features that make the environment more readily applicable to engineering design tasks. Many of these features take the form of predefined primitives that can be applied to subnodes to provide certain functions, discussed below. The prototype environment makes each of these available through a button that inserts a new subnode in the focused model with the corresponding primitive as applicand.

Table 5.2 shows a number of predefined primitives used for performing operations on lists: these kinds of operations are standard elements of most functional programming languages, and parallel some of the capabilities in the spreadsheet environment used for comparison in the case study. The second column of the table gives intuitive mathematical representations of the inputs that would be passed to the sinks on the left-hand sides of subnodes applying these primitives, while the third column gives the corresponding outputs that would be obtained from their sources. The last column of the table indicates how they may be implemented in MDPL; these encodings bear some similarities to typical implementations in symbolic functional languages.

Another predefined primitive provides a simple plotting function that accepts an MDPL model and plots its first output over a supplied range of values for its first input. To simplify the tasks in the case study, the version of the plotting function in the prototype environment generates “log-log” plots that are logarithmically scaled in both variables. With appropriate inputs, the environment normalizes these primitives to a graphical form displaying the plot, which appears like any other primitive in the MDPL user interface.

Finally, the prototype environment provides primitives representing some basic mathematical operations familiar in any common programming language. These are indicated with their usual symbols. Note that the addition and multiplication primitives used as applicands can accept any number of arguments and return the sum or product of all. Subtraction and division are carried out by first multiplying one of the arguments by -1 or raising it to the -1 power, then applying the addition or multiplication operator, respectively.

The acyclic digraph representation used for MDPL models offers much more compact representations of certain mathematical functions than the traditional expression trees of symbolic functional languages. For instance, Figure 5-1 shows an MDPL model representing the three roots of a cubic polynomial equation in terms of primitive mathematical operators, while Figure 5-2 shows the same three roots in the more restrictive expression tree form, such as would be returned by a computer algebra system. The prototype environment uses precomputed versions of compact representations like that in Figure 5-1 when supplying submodels for univariate polynomial roots in the partial inversion transformation, which substantially reduces the size of the resulting models.

Compact visual representations of MDPL models like the ones in Chapter 3 are extremely challenging to draw algorithmically, and such layout problems constitute a research topic in themselves. The prototype environment uses a hybrid algorithm for drawing MDPL models, which uses poly-

Table 5.2: Definitions and interpretations of MDPL primitives for list operations available in the prototype environment

Name	Inputs	Outputs	Label	Definition
Append	$[a_1, \dots, a_n], a_{n+1}$	$[a_1, \dots, a_n, a_{n+1}]$		
Expend	$[a_1, \dots, a_n]$	$[a_1, \dots, a_{n-1}, a_n]$		
List	a_1, a_2, \dots, a_n	$[a_1, \dots, a_n]^1$		
Apply	$f, [a_1, \dots, a_n]$	$f(a_1, \dots, a_n)$		
Map	$f, [a_1, \dots, a_n]$	$[f(a_1), \dots, f(a_n)]$		
Defined?	a, s, t	$\begin{cases} t & a = \emptyset \\ s & a \neq \emptyset \end{cases}$		

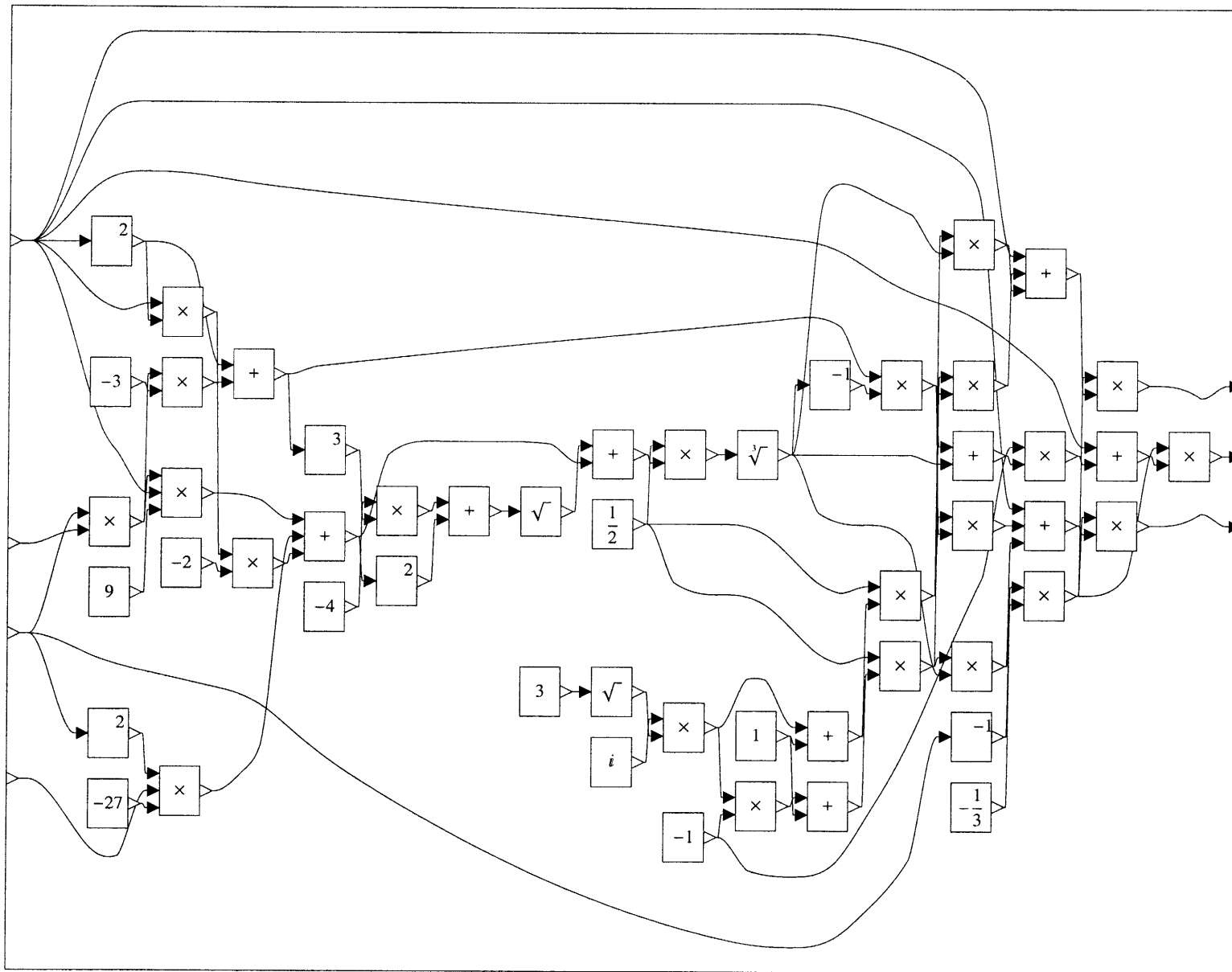


Figure 5-1: MDPL model computing the three roots of the cubic polynomial $ax^3 + bx^2 + cx + d$ over the complex numbers from the coefficients b , c , a , and d (in order from top to bottom), where i denotes the complex unit $\sqrt{-1}$ and a superscript n denotes taking the n th power

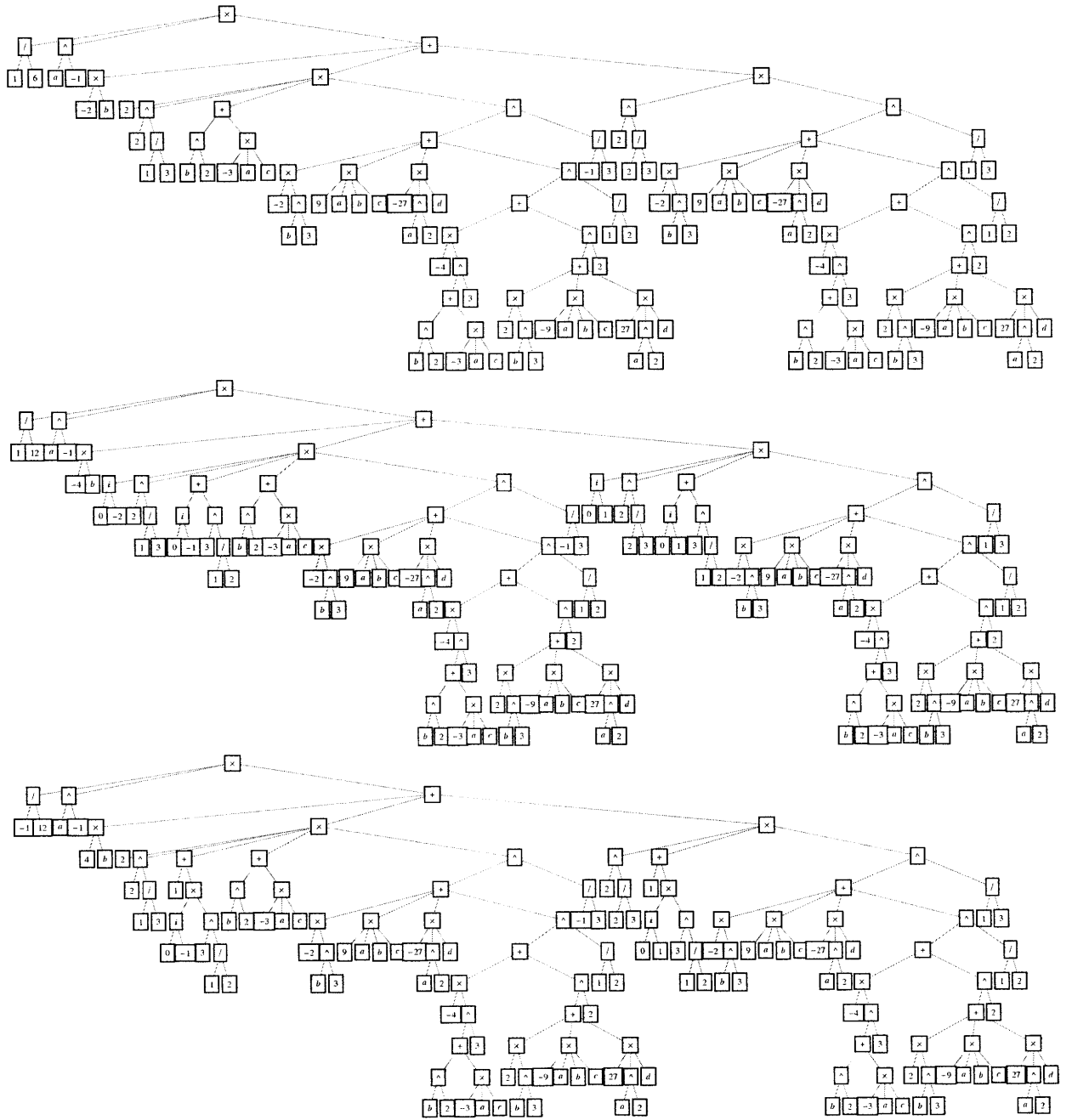


Figure 5-2: Expression trees corresponding to the three roots of the cubic polynomial ax^3+bx^2+cx+d over the complex numbers in terms of coefficients $a, b, c,$ and $d,$ where i denotes the complex combination $u + iv$ of its two subelements u and v

nomial interpolation to merge global positioning and routing of nodes and arrows generated by layered digraph drawing algorithms [29] with local positioning and routing rules around subnodes determined by aesthetic considerations. Figure 5-1 shows an example of a layout generated by this approach. While leaving much room for improvement, this method for automated layout proved adequate for practical use in the case study of §5.2.

5.2 Case Study: Space Telescope Architecture Analysis

The case study described here derives from an engineering design exercise given to students in the 16.89 graduate space systems engineering course at MIT in 2007 [20]. While small enough to be analyzed in detail, it is sufficiently complicated to benefit substantially from the capabilities enabled by MDPL. Moreover, it reflects a real-world conceptual design problem relevant to major decisions in NASA's space exploration program. The results of this case study indicate some key strengths of MDPL, and suggest that these strengths will become increasingly important as the scale and complexity of design tasks increases.

The case study assesses the benefits of using an MDPL-based analysis tool on the performance of a design task. The design task analyzes rough trade-offs in performance and cost for astronomical telescopes observing different wavelengths (radio frequency, infrared, and visible) at different locations (on the Earth's surface, on the Moon's surface, or in space at the Earth-Sun Lagrange point ESL_2). This type of analysis may impact the architecture of the next major space telescope after the James Webb Space Telescope (JWST), and potentially the architecture of the lunar exploration program, though of course a rigorous analysis must account for differences in key technologies and noise environments not captured in this design task.

To experimentally evaluate the practical impact of using MDPL, the author randomly assigned nine volunteers from the 16.89 class to one of two groups. All participants were given the document in §5.2.1 describing the design task and including the relevant domain knowledge necessary to generate solutions. One group of five participants was assigned to complete the design task using the Microsoft[®] Excel[®] spreadsheet environment, which is currently used in the leading integrated design centers of the aerospace industry [12]. The other group of four participants was assigned to complete the design task using the prototype MDPL environment described in §5.1. Participants in the first group were all familiar with Excel, and were given a brief review to confirm their familiarity with relevant features; participants in the second group were given a brief introduction to the MDPL environment covering the features described earlier.

The design task used in the experiment differed from that given in the class assignment in three key ways. First, participants were previously familiar with the overall task from their experience in the 16.89 class, as one might similarly expect of experts addressing a design problem in their field.

Second, the version in the experiment provided the necessary domain knowledge in a readily usable form, as one might similarly expect a professional engineer to have relevant references close at hand. Finally, the version in the experiment requested specific analytical results different from those in the original exercise, based on the same underlying relationships between design parameters.

The first two changes were made to better reflect the characteristics of a professional engineering environment, and to allow the task to be performed in a single 90-minute session with each participant. The final change was made to limit the effect of participants' previous efforts in the class exercise on their performance in the experiment, and to better explore some of the capabilities enabled by MDPL.

The following section replicates the document given to experiment participants describing the design task. To summarize:

1. Part I presents a mathematical model of an astronomical telescope and a family of initial design parameters as initial inputs to the model, and asks participants to calculate the values of the remaining design parameters according to the model.³ The former are among the key physical and environment parameters of an astronomical telescope, while the latter are among the key performance parameters. This calculation drives the participant to effectively implement the model described by the domain knowledge provided.
2. Part II asks participants to assess two bivariate trade-offs between two different pairs of design parameters, while holding certain system parameters constant and allowing other design parameters to vary. The first of these trades system cost against integration time; the latter determines the number of targets observable in a given period of operations. The second of these trades integration time against signal-to-noise ratio for the extraterrestrial locations within a mass cap imposed by the largest available space launch vehicle. These roughly describe the major trade-offs facing designers of a new state-of-the-art single-aperture telescope, and drive the participant to conduct comparative analysis using the model under different constraints.
3. Part III asks participants to consider a modified architecture employing two single-aperture telescopes separated between one terrestrial and one extraterrestrial location for use as an interferometer. It proposes a crude model for evaluating the performance of the two options for this architecture, to illuminate the competing factors in the model, and drive the participant to reuse elements of the original model in a new way.

Note that participants were given Part II only after completing Part I, and Part III only after completing Part II: as in real engineering design work, this prevented them from anticipating the specific questions they might later wish to address when constructing their initial implementation.

³Note: this usage of "model" is distinct from "MDPL model" — we use the latter when necessary to avoid ambiguity.

Participants who could not complete all parts of the design task were asked to spend ~5 minutes at the end to speculate on the approach they would pursue with more time to complete the task using their environment.

5.2.1 Design Task

This design task is divided into 3 parts. You will receive each part of the task upon completion of the previous part. This is intended not to frustrate you, but rather to represent the unanticipated changes in direction often encountered in real design studies.

Since time is a major factor in the assessment of results from this experiment, you are asked to complete the tasks in each part as quickly as you can. You should feel free to ask any questions of the experimenter, though the experimenter may choose not to answer some questions.

Part I: Model Construction This part asks you to construct a quantitative model representing a family of astronomical telescopes using the modeling environment.

Please use only the parameters and relationships provided below to construct your model. Do not worry about the underlying correctness of the model or the information given.

Table 5.4 describes the parameters whose values are given as inputs in Part I. Table 5.6 describes the parameters you will need to calculate using the model.

Your model should cover telescopes at three frequency bands (RF, IR, VIS) and three locations (Earth, ESL₂, Moon). Some of the input parameters may vary according to frequency band and/or location, as indicated in Table 5.4.

The final outputs of your model in Part I should be angular resolution (θ_r), signal-to-noise ratio (SNR), and total cost (C_T).

Table 5.4: Parameters with given values in Part I

Parameter	Symbol	Value	Units			
instantaneous field of view	$IFOV$	7.272×10^{-5}	rad			
target distance	r	9.257×10^{19}	m			
operating wavelength	λ	RF	2.1×10^{-1}	m		
		IR	1.0×10^{-5}			
		VIS	5.0×10^{-7}			
detector pixel width	d	3.0×10^{-5}	m			
quality factor	Q	1.1	–			
Rayleigh factor	ρ	1.220	–			
Boltzmann constant	k	1.381×10^{-23}	J/K			
target blackbody temperature	T	5785	K			
Planck constant	h	6.626×10^{-34}	J/s			
light speed	c	2.998×10^8	m/s			
atmospheric transmission factor	τ_λ		Earth	ESL ₂	Moon	–
		RF	1.00	1.00	1.00	
		IR	0.80	1.00	1.00	
		VIS	0.98	1.00	1.00	
bandwidth	$\Delta\lambda$	2.0×10^{-6}	m			
target radius	R	6.96×10^8	m			
net velocity change for orbital maneuvers	ΔV	Earth	ESL ₂	Moon	m/s	
		0	12 800	15 400		
Earth surface gravitational acceleration	g	9.8	m/s ²			
rocket fuel specific impulse	I_{sp}	450	s			
specific transport cost	c_t	1163	\$/kg			
telescope mass scale factor	m_s	1175	kg/m ²			
quantum efficiency	QE	0.5	–			
readout noise electron count	N_r	25	–			
optical transmission factor	τ_0	0.75	–			
observation integration time	T_i	30	s			
primary aperture cost scale factor	α		Earth	ESL ₂	Moon	\$/m ³
		RF	50 000	500 000	100 000	
		IR	500 000	1 000 000	1 000 000	
		96 VIS	500 000	1 000 000	1 000 000	

Table 5.6: Parameters with computed values in Part I

Parameter	Symbol	Expression	Units
target pixel resolution	Y	$= IFOV \cdot r$	m
focal length	f	$= \frac{r \cdot d}{Y}$	m
primary aperture diameter	D	$= \frac{2 \cdot \rho \cdot \lambda \cdot Q \cdot f}{d}$	m
angular resolution	θ_r	$= \frac{\rho \cdot \lambda}{D}$	rad
telescope mass	m_t	$= m_s \cdot D^2$	kg
transport mass	m_i	$= m_t \cdot \exp\left(\frac{\Delta V}{g \cdot I_{sp}}\right)$	kg
primary aperture cost	C_D	$= \alpha \cdot D^3$	\$
total cost	C_T	$= C_D + c_t \cdot m_i$	\$
target spectral irradiance	H_λ	$= \frac{2\pi \cdot h \cdot c^2}{\lambda^5} \cdot \frac{1}{\exp\left(\frac{c \cdot h}{k \cdot T \cdot \lambda}\right) - 1}$	W/m ³
target intensity	L	$= \frac{1}{4} \cdot R^2 \cdot H_\lambda \cdot \tau_\lambda \cdot \Delta\lambda$	W/sr
sensor input power	P_{in}	$= \pi \cdot \left(\frac{D}{2 \cdot \tau}\right)^2 \cdot L$	W
integrated energy	H_i	$= P_{in} \cdot \tau_0 \cdot T_i$	J
available photon count	N_p	$= \frac{H_i \cdot \lambda}{h \cdot c}$	-
signal-to-noise ratio	SNR	$= \frac{N_p \cdot QE}{\sqrt{N_p^2 + N_p \cdot QE}}$	-

Tasks:

- I.1. Using the modeling environment, build a model capturing the parameter values and relationships indicated below. Your model should be capable of accepting any reasonable value for any of the input parameters.
- I.2. For each frequency band and location, compute the angular resolution (θ_r), signal-to-noise ratio (SNR), and total cost (C_T) for the telescope system described by your model for the given input values. Record these values in the form of Table 5.8.

Table 5.8: Results from the model in Part I

angular resolution θ_r [rad]				signal-to-noise ratio SNR [-]				total cost C_T [\$]			
	Earth	ESL ₂	Moon		Earth	ESL ₂	Moon		Earth	ESL ₂	Moon
RF				RF				RF			
IR				IR				IR			
VIS				VIS				VIS			

Part II: Trade Analysis This part asks you to manipulate your model to answer certain questions of interest to telescope designers. Please do each of the following for each frequency band and each location where applicable.

Tasks:

II.1. Allow the instantaneous field of view (*IFOV*) and the single-observation integration time (T_i) to vary. Assume a fixed signal-to-noise ratio (*SNR*) of 10. Create a plot of the single-observation integration time (T_i) on the vertical axis versus total cost (C_T) on the horizontal axis, with the total cost (C_T) on a logarithmic scale from $\$10^6$ to $\$10^{10}$, with values at successive multiples of 10.

II.2. Start from the model you constructed in the previous step. Now consider only the space locations (ESL₂ and Moon). Allow the operating wavelength (λ) to vary, and assume the fixed values $\alpha = 1\,000\,000$ $\$/\text{m}^3$ and $\tau_\lambda = 1.00$. Assume that the transport mass (m_i) is fixed at 60 000 kg by the launch vehicle. Create a plot of the total cost (C_T) on the vertical axis versus single-observation integration time (T_i) on the horizontal axis, with the single-observation integration time (T_i) on a logarithmic scale from 10^0 to 10^4 seconds, with values at successive multiples of 10.

Part III: Model Extension This part asks you to extend your model to compare options for a modified architecture for radioastronomical observations. Instead of a single-aperture radio telescope, the modified architecture employs two single-aperture radio telescopes in different locations, combining their observations to perform very long baseline interferometry (VLBI). In order to achieve a long baseline for a low cost, the two telescopes are to be located either at the Earth and the Moon, or at the Earth and ESL₂. As a very crude starting point, model the two-telescope system with the following assumptions:

- the telescope performance model from Parts I-II applies individually to each of the two telescopes in the system
- the two telescopes both operate at radio frequency (RF) wavelengths
- the net cost of the system is the sum of the costs of the individual telescopes, $C_T = C_{T_1} + C_{T_2}$
- the net angular resolution of the system is that obtained by an individual telescope with a diameter equal to the baseline l with values in Table 5.9, $\theta_r = \frac{\rho \cdot \lambda}{l}$
- the net signal-to-noise ratio is the ratio of the sum of the signal photon counts from both telescopes to the sum of the noise photon counts from both telescopes, $SNR = \frac{(N_{p_1} + N_{p_2}) \cdot QE}{\sqrt{2 \cdot N_r^2 + (N_{p_1} + N_{p_2}) \cdot QE}}$

Table 5.9: Additional parameter values in Part III

Parameter	Symbol	Value		Units
baseline (synthetic aperture diameter)	l	Earth-ESL ₂	Earth-Moon	m
		1 500 000	380 000	

Please do each of the following for each of the two system options.

Tasks:

- III.1. Let each telescope have the initial values given for all parameters listed in Table 5.4. Modify the original model from Part I to calculate the net system angular resolution (θ_r), signal-to-noise ratio (SNR), and total cost (C_T) as described above. Record these values in the form of Table 5.11.
- III.2. Start from the model you constructed in the previous step. Allow the instantaneous field of view of both telescopes ($IFOV_1, IFOV_2$) to vary. Assume a fixed cost (C_T) equal to the value you found in the previous step. Create a plot of the diameter of the extraterrestrial telescope (D_2) on the vertical axis versus the diameter of the terrestrial telescope (D_1) on the horizontal axis, with the diameter D_1 on a logarithmic scale from 10^{-1} to 10^3 m, with values at successive multiples of 10.

Table 5.11: Results from the model in Part III

angular resolution θ_r [rad]		signal-to-noise ratio SNR [-]		total cost C_T [\$]	
Earth-ESL ₂	Earth-Moon	Earth-ESL ₂	Earth-Moon	Earth-ESL ₂	Earth-Moon

5.2.2 Task Analysis

Before discussing the results of the experiment in §5.2.3, we describe here the reasoning process required to complete the above design task, and the basic steps of its realization in each experimental environment. Both environments offer considerable flexibility in implementation: here we discuss only one approach specific to each environment, while §5.2.3 discusses alternative approaches observed in the experiment.

Part I: Model Construction Figure 5-3 shows a visual representation of the binary matrix whose rows correspond to values provided in Table 5.4 and equations provided in Table 5.6, whose

columns correspond to the variables in the model, and whose entries indicate which variables relate to a particular value or equation. The information represented here is immediately apparent to the engineer from looking at Tables 5.4 & 5.6 and the symbols contained in each equation, and the engineer need not explicitly construct this matrix.

In the block triangular decomposition of the above matrix, all row- and column-subsets consist of a single row and column (Figure 5-4). This indicates that values for all variables can be found by finding an equation with a single variable whose value is unknown, computing the value of that variable by solving for it in the equation, and repeating this procedure until values have been computed for all variables. The order in which equations are used is given by the ordering of rows in the block triangular decomposition. In this particular case, no equation solving is required, since in each step the unknown variable is already isolated on one side of the equation.

The above properties may not be immediately apparent to the engineer, however, they may suspect as much from their prior experience with the model used in the design task. Models for conceptual design are often initially formulated with these properties, since preserving them at each step in the construction of the model provides a simple check that the model is well determined. In engineering design, models in this form often accept physical design parameters as inputs and yield performance design parameters as outputs. A model satisfying these properties can be used fairly easily to calculate its output parameters, since individual parameters can be computed one by one using individual equations. This makes the implementation of such a model in the spreadsheet environment straightforward, where each equation is entered as a one-line formula (Figure 5-5). In the corresponding MDPL model generated from the model equations (Figure 5-6), it is reflected in that each subnode, representing one equation, has a single source (Figure 5-7).

To incorporate different families of initial values for the variable parameters of telescope wavelength and location, each environment offers multiple approaches. In the spreadsheet environment, with two variable parameters, the design parameters dependent on one variable are allocated multiple corresponding horizontally adjacent cells, and those dependent on the other variable are allocated multiple corresponding vertically adjacent cells, while those dependent on both variables are allocated an array of cells spanning multiple rows and columns (Figure 5-8). An array for a parameter with given values is filled with the corresponding values, while an array for a parameter with computed values is filled by modifying the original formula to identify constituent parameters' non-dependencies on either variable (using the '\$' character in Excel®), and then copying the resulting formula across all cells in the array. This approach requires modifying multiple references in nearly every formula of the model, and copying nearly every formula over multiple cells (which must be redone whenever formulas are changed later on). This type of approach fails in the case of more than two variable parameters, and generalizing it requires even more extensive manual changes to references in cell formulas.

	<i>c</i>	<i>d</i>	<i>D</i>	<i>f</i>	<i>g</i>	<i>h</i>	IFOV	<i>k</i>	<i>L</i>	<i>Q</i>	QE	<i>r</i>	<i>R</i>	SNR	<i>T</i>	<i>Y</i>	α	ΔV	$\Delta\lambda$	λ	ρ	<i>c_t</i>	<i>C_D</i>	<i>C_T</i>	<i>H_f</i>	<i>H_s</i>	<i>m_i</i>	<i>m_e</i>	<i>m_t</i>	<i>N_p</i>	<i>N_t</i>	<i>P_{in}</i>	<i>T_i</i>	θ_i	<i>I_{sp}</i>	τ_0	τ_λ				
<i>c</i>	■	
<i>d</i>	.	■	
<i>g</i>	■	
<i>h</i>	■	
IFOV	■	
<i>k</i>	■	
<i>Q</i>	■	
QE	■	
<i>r</i>	■	
<i>R</i>	■	
<i>T</i>	■	
α	■	
ΔV	■	
$\Delta\lambda$	■	
λ	■	
ρ	■	
<i>c_t</i>	■
<i>m_e</i>	■
<i>N_i</i>	■
<i>T_i</i>	■
<i>I_{sp}</i>	■
τ_0	■
τ_λ	■
$Y = \text{IFOV } r$	■	■
$f = \frac{d_i}{Y}$.	■	.	■	■
$D = \frac{2 f \theta \lambda D}{d}$.	■	■	■	■	■	■	
$\theta_i = \frac{\lambda D}{D}$.	.	■	■	■	
$m_e = D^2 m_e$.	.	■	■	■	
$m_i = e^{f/\lambda} m_e$	■	■	■	■	
$C_D = D^3 \alpha$.	.	■	■	■	
$C_T = C_D + c_t m_i$	■	■	■	
$H_s = \frac{2 c_t h \rho}{1 + e^{-\rho \lambda}}$	■	.	.	.	■	.	.	■	■	■	
$L = \frac{1}{4} R^2 \Delta \lambda H_s \tau_0$	■	.	.	.	■	■	■	.
$P_{in} = \frac{P \lambda D}{4 \lambda}$.	.	■	■	.	.	■	
$H_i = P_{in} T_i \tau_0$	■	■	
$N_p = \frac{\lambda H_i}{c h}$	■	.	.	.	■	■	.	.	.	■	
$\text{SNR} = \frac{\text{OE } N_i}{\sqrt{N_i \cdot \text{OE } N_i}}$	■	.	■	■	■

Figure 5-3: Binary matrix relating values and equations to variables in the space telescope model, where $\cdot = 0$ and $\blacksquare = 1$

	c	d	r	IFOV	Y	f	Q	λ	ρ	D	g	h	k	R	$\Delta\lambda$	T	H_λ	τ_λ	L	QE	P_{in}	T_f	τ_0	H_f	N_p	N_t	SNR	α	ΔV	C_t	C_D	m_g	m_t	I_{sp}	m_i	C_T	θ_i			
c	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
d	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
r	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
IFOV	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$Y = IFOV r$	*	*	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$f = \frac{d_i}{Y}$	*	■	■	*	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Q	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
λ	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
ρ	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$D = \frac{2f\theta\lambda\rho}{d}$	*	■	*	*	*	■	■	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
g	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
h	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
k	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
R	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$\Delta\lambda$	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
T	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$H_\lambda = \frac{2\sigma_{h\nu}}{\{1 + \frac{h\nu}{kT}\}^2}$	■	*	*	*	*	*	*	■	*	*	*	■	■	*	*	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
τ_λ	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$L = \frac{1}{4} R^2 \Delta\lambda H_\lambda \tau_\lambda$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	■	*	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
QE	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$P_{in} = \frac{P^i L n}{4 r^2}$	*	*	■	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
T_f	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
τ_0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$H_f = P_{in} T_f \tau_0$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$N_p = \frac{\lambda N_s}{c h}$	■	*	*	*	*	*	*	■	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
N_t	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$SNR = \frac{OE N_t}{\sqrt{N_s \cdot OE N_s}}$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*
α	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
ΔV	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
C_t	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$C_D = D^3 \alpha$	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
m_g	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$m_t = D^2 m_g$	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
I_{sp}	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$m_i = e^{\frac{-\Delta V}{kT}} m_t$	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$C_T = C_D + c_t m_i$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	■
$\theta_i = \frac{\lambda D}{D}$	*	*	*	*	*	*	*	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■

Figure 5-4: Block triangular decomposition of the matrix in Figure 5-3, where lines separate the corresponding row- and column-subsets

	A	B	C	D	E	F	G	H
1	IFOV	7.272E-005	Y	6.732E+015				
2	r	9.257E+019	f	4.125E-001				
3	lambda	2.100E-001	D	7.751E+003				
4		1.000E-005	theta_r	3.305E-005				
5		5.000E-007	m_l	7.059E+010				
6	d	3.000E-005	m_j	7.059E+010				
7	Q	1.100E+000	C_D	2.328E+016				
8	rho	1.220E+000	C_T	2.336E+016				
9	k	1.381E-023	H_lambda	=2*PI()*B11*B12^2/B3^5/(EXP(B12*B11/(B9*B10*B3))-1)				
10	T	5.785E+003	L	1.874E+004				
11	h	6.626E-034	P_in	1.032E-028				
12	c	2.998E+008	H_l	2.322E-027				
13	tau_lambda	1.000E+000	N_p	2.455E-003				
14	Delta lambda	2.000E-006	SNR	4.909E-005				
15	R	6.960E+008						
16	Delta V	0.000E+000						
17	g	9.807E+000						
18	I_sp	4.500E+002						
19	c_f	1.163E+003						
20	m_s	1.175E+003						
21	QE	5.000E-001						
22	N_r	2.500E+001						
23	tau_0	7.500E-001						
24	T_l	3.000E+001						
25	alpha	5.000E+004						

Figure 5-5: Spreadsheet implementation of the initial telescope model in Part I, where the highlighted formula corresponds to the equation $H_{\lambda} = \frac{2\pi \cdot h \cdot c^2}{\lambda^5} \cdot \frac{1}{\exp(\frac{c \cdot h}{k \cdot T \cdot \lambda}) - 1}$, and values shown are for the initial Earth-based RF design

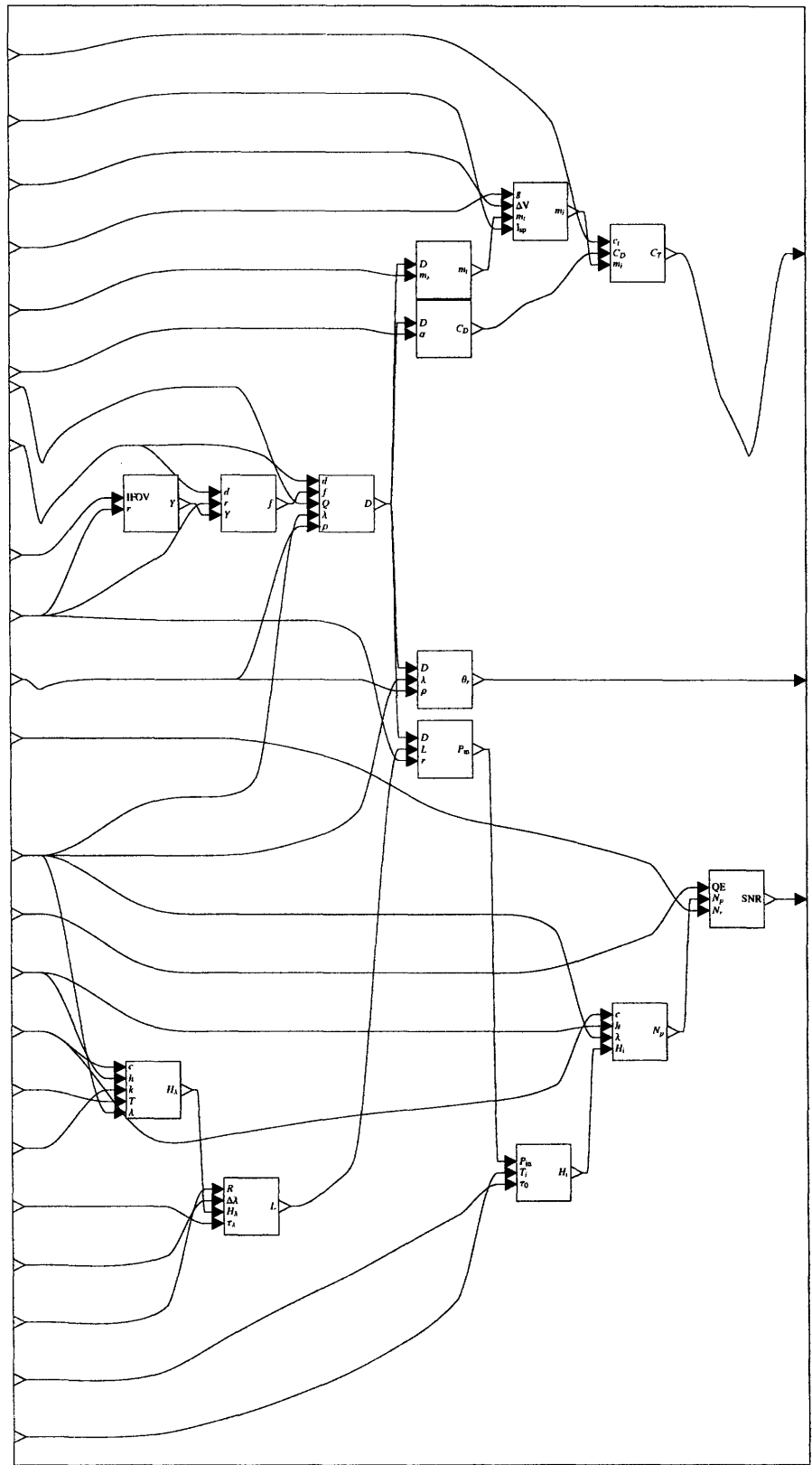


Figure 5-6: MDPL implementation of the initial telescope model in Part I, where subnodes shown have submodel applicands implementing equations in the model corresponding to their labels

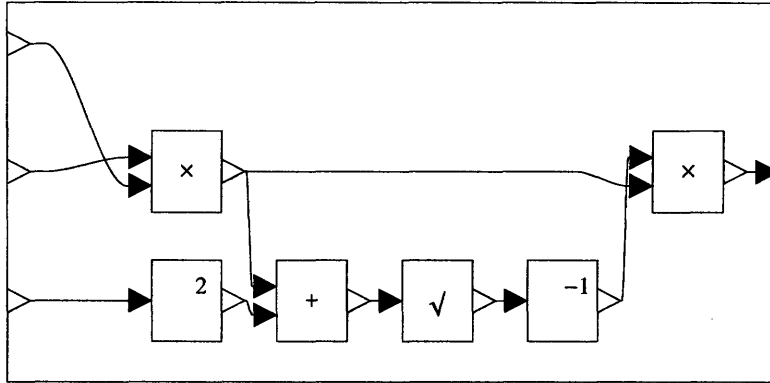


Figure 5-7: MDPL submodel applicand of the rightmost subnode in Figure 5-6, corresponding to the equation $SNR = \frac{N_p \cdot QE}{\sqrt{N_p^2 + N_p \cdot QE}}$

The MDPL environment offers several different approaches: the one shown in Figure 5-9 uses some of the built-in primitives described in §5.1 that provide functionality for handling arrays. (Note that in order to display the submodel applicand of the uppermost subnode in this model, Figure 5-9 is a drawing rather than the visualization provided by the prototype GUI.) After values for the parameters in Table 5.4 have been entered as primitives, those except ΔV , α , λ , and τ_λ are linked to the corresponding sinks on the model of Figure 5-6. Applying EMS to the result creates the subnode in the upper right of Figure 5-9. The remainder of the model is constructed by creating new subnodes via EMS and using RSR and RSP to build up the model as shown. Though the result is somewhat messy, the number of user interface operations is fairly small. Note that many operations are saved by defining the applicands of the second and third large subnodes using the first one. This approach requires forming several new subnodes, arguments, and applicands, but does not require any modifications to the original telescope model. Moreover, this type of approach extends naturally to handle more variable parameters with minimal additional effort.

The values of the output design parameters for Part I can be read directly from the corresponding cells in the spreadsheet environment, or from the neatly arranged lists of numeric primitives in the normal form model produced by the MDPL environment. The latter representation is adequate for this case study, though future MDPL environments may offer a special compact display format to better facilitate examining large results in the form of arrays.

Part II: Trade Analysis In Part II.1, the model must be modified so that SNR and C_T are input parameters while T_i is an output parameter. In the MDPL environment, this is done by applying the partial inversion transformation, using a permutation that swaps SNR and C_T with $IFOV$ and T_i . In the spreadsheet environment, it requires considerable effort.

Figure 5-10 shows the block triangular decomposition for this case, in the style of Figure 5-4. This shows that the row- and column-subsets no longer consist strictly of single rows and single columns:

	A	B	C	D	E	F	G	H
1	IFOV	7.272E-005			Y	6.732E+015		
2	r	9.257E+019			f	4.125E-001		
3	lambda	2.100E-001			D	7.751E+003		
4		1.000E-005				3.691E-001		
5		5.000E-007				1.845E-002		
6	d	3.000E-005			theta_r	3.305E-005		
7	Q	1.100E+000				3.305E-005		
8	rho	1.220E+000				3.305E-005		
9	k	1.381E-023			m_t	7.059E+010		
10	T	5.785E+003				1.601E+002		
11	h	6.626E-034				4.002E-001		
12	c	2.998E+008				m_j = \$F9*EXP(B\$18/(\$B\$19*\$B\$20))		
13	tau_lambda	1.000E+000	1.000E+000	1.000E+000		1.601E+002	2.910E+003	5.246E+003
14		8.000E-001	1.000E+000	1.000E+000		4.002E-001	7.276E+000	1.311E+001
15		9.800E-001	1.000E+000	1.000E+000	C_D	2.328E+016	2.328E+017	4.656E+016
16	Delta lambda	2.000E-006				2.514E+004	5.028E+004	5.028E+004
17	R	6.960E+008				3.142E+000	6.285E+000	6.285E+000
18	Delta V	0.000E+000	1.280E+004	1.540E+004	C_T	2.336E+016	2.343E+017	4.925E+016
19	g	9.807E+000				2.113E+005	3.435E+006	6.151E+006
20	l_sp	4.500E+002				4.685E+002	8.468E+003	1.526E+004
21	c_t	1.163E+003			H_lambda	7.738E-008		
22	m_s	1.175E+003				1.326E+010		
23	QE	5.000E-001				8.347E+013		
24	N_r	2.500E+001			L	1.874E+004	1.874E+004	1.874E+004
25	tau_0	7.500E-001				2.568E+021	3.211E+021	3.211E+021
26	T_j	3.000E+001				1.981E+025	2.022E+025	2.022E+025
27	alpha	5.000E+004	5.000E+005	1.000E+005	P_in	1.032E-028	1.032E-028	1.032E-028
28		5.000E+005	1.000E+006	1.000E+006		3.207E-020	4.009E-020	4.009E-020
29		5.000E+005	1.000E+006	1.000E+006		6.184E-019	6.310E-019	6.310E-019
30					H_j	2.322E-027	2.322E-027	2.322E-027
31						7.215E-019	9.019E-019	9.019E-019
32						1.391E-017	1.420E-017	1.420E-017
33					N_p	2.455E-003	2.455E-003	2.455E-003
34						3.632E+001	4.540E+001	4.540E+001
35						3.502E+001	3.574E+001	3.574E+001
36					SNR	4.909E-005	4.909E-005	4.909E-005
37						7.161E-001	8.920E-001	8.920E-001
38						6.909E-001	7.048E-001	7.048E-001

Figure 5-8: Spreadsheet expanding the implementation of Figure 5-5 over multiple wavelengths and locations

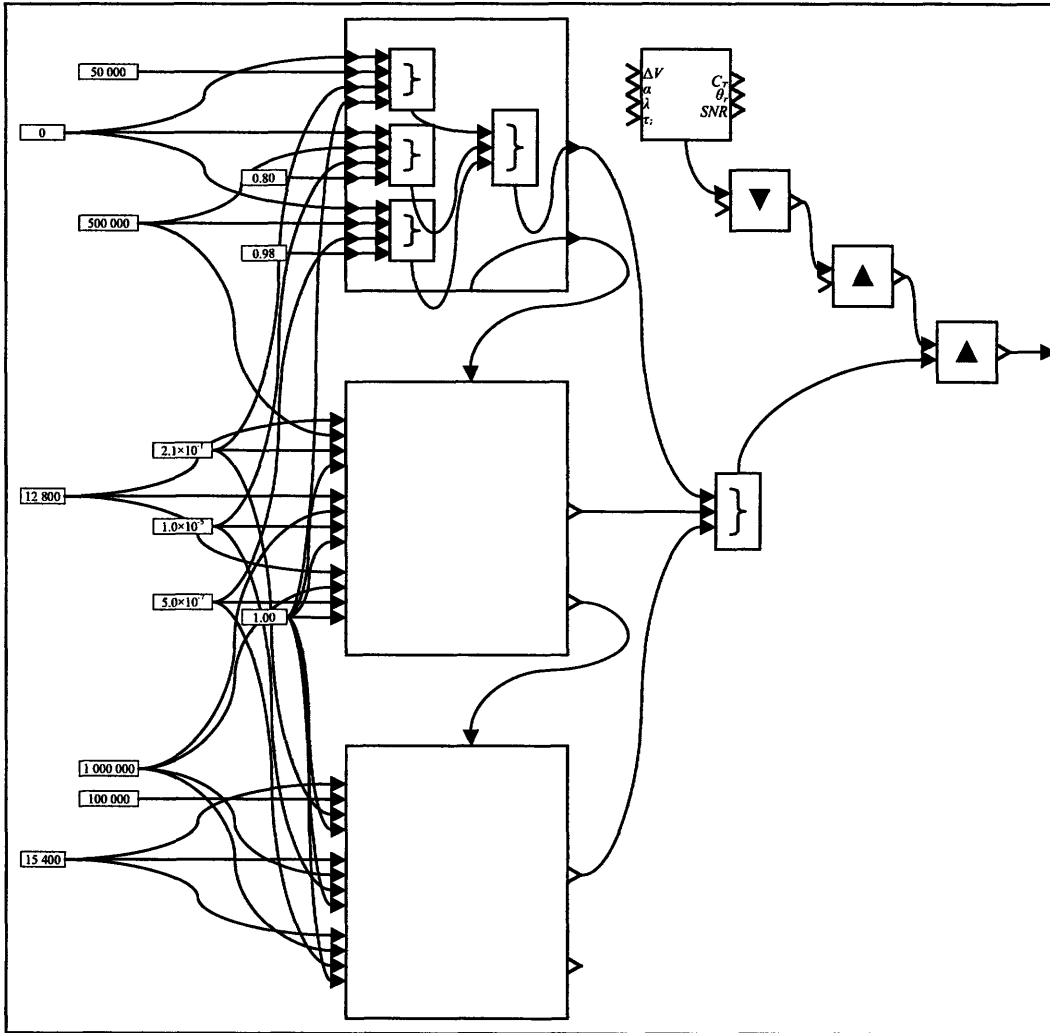


Figure 5-9: MDPL model applying the model in Figure 5-6 to multiple wavelengths and locations

one pair of corresponding row- and column-subsets involve four rows and four columns. This indicates that the corresponding equations must be solved simultaneously to express the parameters C_D , D , m_i , and m_t in terms of the parameters α , c_t , C_T , g , I_{sp} , m_s , and ΔV . In the spreadsheet environment, with no built-in capacity to analyze the model structure, determining this fact requires careful analysis.

The solution of the equations poses additional challenges to the engineer using the spreadsheet environment. Computing the solution by hand or (more likely) with a computer algebra system yields a very large result: for instance, the first two solutions for C_D alone, even after algebraic simplification, span nearly a page below; the third solution is similar in form to the second. For each of the parameters C_D , D , m_i , and m_t , the engineer must then check which is the desired solution by substituting in the initial values for the known parameters from the previous implementation of the model. (The numerical solver in the Excel[®] software does not consistently converge to the correct solution, and cannot be reliably used for this task.) Finally, the engineer must encode these new formulas for each of the parameters into the spreadsheet. Here the engineer must decide whether to overwrite the existing formulas of the original model to maintain the same layout (requiring additional effort to recover them for Part II.2 and Part III), to do the same after copying the original model (leading to the aforementioned pitfalls of copying), or to create new cells for the modified formulas (creating a complicated branched layout less amenable to further manipulations).

$$\begin{aligned}
C_D = C_T - \frac{1}{18\alpha^2} e^{\frac{\Delta V}{g I_{sp}}} c_t m_s & \left(6e^{\frac{2\Delta V}{g I_{sp}}} c_t^2 m_s^2 \right. \\
& + \frac{2^{5/3} e^{\frac{4\Delta V}{g I_{sp}}} c_t^4 m_s^4}{\left(-2e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3 + 27\alpha^2 C_T + 3\sqrt{3} \sqrt{\alpha^2 C_T \left(27\alpha^2 C_T - 4e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3 \right)} \right)^{2/3}} \\
& - \frac{4\sqrt[3]{2} e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3}{\sqrt[3]{-2e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3 + 27\alpha^2 C_T + 3\sqrt{3} \sqrt{\alpha^2 C_T \left(27\alpha^2 C_T - 4e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3 \right)}}} \\
& - 2^{5/3} e^{\frac{\Delta V}{g I_{sp}}} c_t m_s \sqrt[3]{-2e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3 + 27\alpha^2 C_T + 3\sqrt{3} \sqrt{\alpha^2 C_T \left(27\alpha^2 C_T - 4e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3 \right)}}} \\
& \left. + \sqrt[3]{2} \left(-2e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3 + 27\alpha^2 C_T + 3\sqrt{3} \sqrt{\alpha^2 C_T \left(27\alpha^2 C_T - 4e^{\frac{3\Delta V}{g I_{sp}}} c_t^3 m_s^3 \right)} \right)^{2/3} \right)
\end{aligned}$$

	c	d	α	c_c	C_T	g	ΔV	m_e	I_{sp}	D	C_D	m_I	m_t	Q	λ	ρ	f	h	r	Y	IPOV	k	R	$\Delta\lambda$	T	H_λ	ϵ_λ	L	QE	SNR	N_i	N_p	H_i	P_{in}	t_0	T_i	Θ_i					
c	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
d	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
α	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
c_c	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
C_T	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
g	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
ΔV	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
m_e	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
I_{sp}	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$C_D = D^3 \alpha$	*	*	■	*	*	*	*	*	*	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$C_T = C_D + c_c m_I$	*	*	*	■	■	*	*	*	*	*	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$m_I = \frac{g}{c} m_t$	*	*	*	*	*	■	■	*	■	*	*	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$m_t = D^2 m_e$	*	*	*	*	*	*	*	■	*	■	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
Q	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
λ	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
ρ	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$D = \frac{2 f \theta_{10}}{d}$	*	■	*	*	*	*	*	*	*	■	*	*	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
h	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Y	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$f = \frac{d t}{Y}$	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$Y = \text{IPOV } r$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
k	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
R	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$\Delta\lambda$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
T	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$H_\lambda = \frac{2 c^2 h n}{(1 + c^2)^{1/2} \lambda}$	■	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	■	*	*	*	*	■	*	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
ϵ_λ	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$L = \frac{1}{4} R^2 \Delta\lambda H_\lambda \epsilon_\lambda$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■	■	■	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*
QE	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
SNR	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
N_i	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$\text{SNR} = \frac{QE N_i}{\sqrt{N_i \cdot QE N_i}}$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$N_p = \frac{\lambda N_i}{c h}$	■	*	*	*	*	*	*	*	*	*	*	*	*	*	■	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
$P_{in} = \frac{P_c \lambda n}{4 r}$	*	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
t_0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$H_i = P_{in} T_i t_0$	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
$\Theta_i = \frac{\lambda \nu}{D}$	*	*	*	*	*	*	*	*	*	■	*	*	*	*	■	■	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	■

Figure 5-10: Block triangular decomposition of the binary matrix relating parameters to values and equations in Part II.1

$$\begin{aligned}
C_D = & \frac{1}{12\alpha^2 \left(-2e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3 + 27\alpha^2 C_T + 3\sqrt{3} \sqrt{\alpha^2 C_T (27\alpha^2 C_T - 4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3)} \right)^{2/3}} \left(2^{5/3} (1 + i\sqrt{3}) e^{\frac{5\Delta V}{gT_{sp}}} c_t^5 m_s^5 \right. \\
& + 12\alpha^2 C_T \left(-2e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3 + 27\alpha^2 C_T + 3\sqrt{3} \sqrt{\alpha^2 C_T (27\alpha^2 C_T - 4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3)} \right)^{2/3} \\
& - 4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3 \left(-2e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3 + 27\alpha^2 C_T + 3\sqrt{3} \sqrt{\alpha^2 C_T (27\alpha^2 C_T - 4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3)} \right)^{2/3} \\
& + 2i (i + \sqrt{3}) e^{\frac{4\Delta V}{gT_{sp}}} c_t^4 m_s^4 \sqrt[3]{-4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3 + 54\alpha^2 C_T + 6\sqrt{3} \sqrt{\alpha^2 C_T (27\alpha^2 C_T - 4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3)}} \\
& + e^{\frac{\Delta V}{gT_{sp}}} c_t m_s \sqrt[3]{-4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3 + 54\alpha^2 C_T + 6\sqrt{3} \sqrt{\alpha^2 C_T (27\alpha^2 C_T - 4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3)}} \\
& \left. \left(9 (1 - i\sqrt{3}) C_T \alpha^2 + (-3i + \sqrt{3}) \sqrt{\alpha^2 C_T (27\alpha^2 C_T - 4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3)} \right) \right. \\
& \left. - 2^{5/3} e^{\frac{2\Delta V}{gT_{sp}}} c_t^2 m_s^2 \left(9 (1 + i\sqrt{3}) C_T \alpha^2 + (3i + \sqrt{3}) \sqrt{\alpha^2 C_T (27\alpha^2 C_T - 4e^{\frac{3\Delta V}{gT_{sp}}} c_t^3 m_s^3)} \right) \right)
\end{aligned}$$

Clearly there is a good deal of structure in the above solutions, which is preserved in the MDPL implementation using the partial inversion transformation, yielding more usable results. The transformation first creates a new subnode via EMS with submodel applicand containing the original subnodes within the component of the block triangular decomposition (Figure 5-11). Applying itself recursively, it then extracts the equations corresponding to the subnodes' submodels, yielding the polynomials

$$\begin{Bmatrix} C_D - D^3\alpha \\ C_D - C_T + c_t m_i \\ m_i - e^{\frac{\Delta V}{gT_{sp}}} m_t \\ m_t - D^2 m_s \end{Bmatrix}$$

for which to find common roots. The recursive application of the partial inversion transformation to the subnodes here essentially amounts to substituting intermediate parameters for constant expressions: in this case, letting $r_m = e^{\frac{\Delta V}{gT_{sp}}}$ in the third equation. The environment then computes

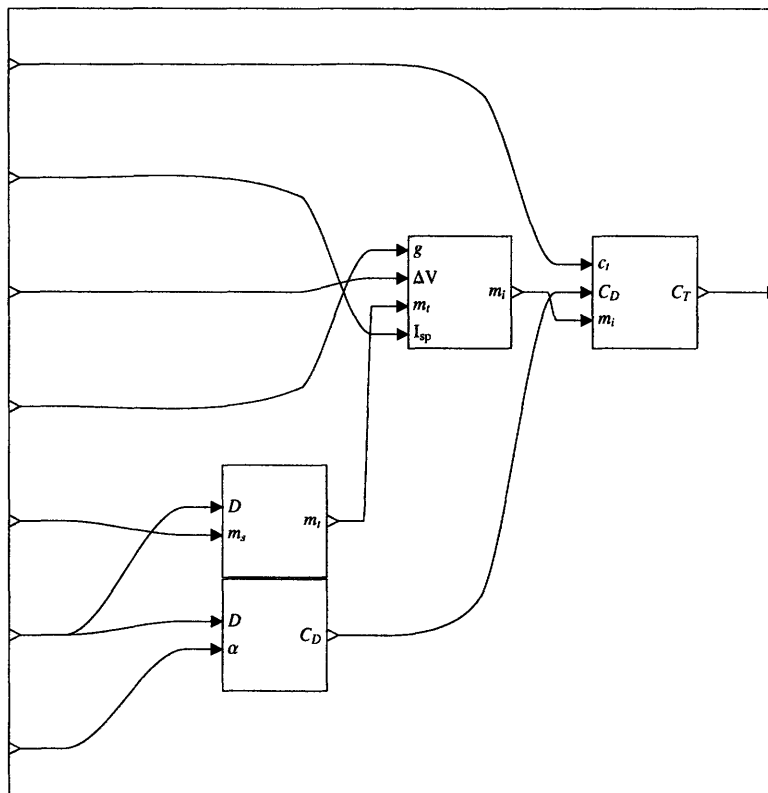


Figure 5-11: Submodel created by the partial inversion transformation containing the subnodes within a single component of the block triangular decomposition in Figure 5-10

the Gröbner basis of the polynomials, yielding the set of polynomials

$$\left\{ \begin{array}{l} m_i - m_t r_m \\ D^2 m_s - m_t \\ C_D - C_T + c_t m_t r_m \\ \alpha D^3 - C_T + c_t m_t r_m \\ -C_T m_s + c_t m_t r_m m_s + D \alpha m_t \\ -D C_T m_s^2 + D c_t m_t r_m m_s^2 + \alpha m_t^2 \\ c_t^2 m_t r_m^2 m_s^3 - c_t C_T r_m m_s^3 + D \alpha C_T m_s^2 - \alpha^2 m_t^2 \\ -C_T^2 m_s^3 - c_t^2 m_t^2 r_m^2 m_s^3 + 2 c_t C_T m_t r_m m_s^3 + \alpha^2 m_t^3 \end{array} \right\}.$$

The last polynomial above contains only the parameter m_t among the parameters to be solved for, so its roots are computed using the compact representation for cubic roots in Figure 5-1 as a submodel. The appropriate root (among three possibilities) is selected by checking against the substitution of the initial values already encoded in the MDPL model. The other parameters m_i , D , and C_D are similarly computed by solving for the roots of the first, second, and third polynomials above, respectively. Though only D is strictly required to provide a complete partial inversion, the prototype implementation includes the other parameters in case the engineer wishes to use them elsewhere: in this sense, information is always preserved at the same level of the submodel hierarchy as it existed before. The resulting subnodes are linked together to form the submodel shown in Figure 5-12, which replaces that in Figure 5-11.

Looking again at the block triangular decomposition in Figure 5-10, we find that several other equations are used to determine variables that are not isolated on their left-hand sides, as was the case in Part I. The recursive application of the partial inversion transformation to the corresponding subnodes of these equations yields subnodes with submodel applicands corresponding to the solved forms of these equations,

$$\begin{aligned} D &= \frac{2 \cdot \rho \cdot \lambda \cdot Q \cdot f}{d} \quad \dashrightarrow \quad f = \frac{D \cdot d}{2 \cdot \rho \cdot \lambda \cdot Q} \\ f &= \frac{r \cdot d}{Y} \quad \dashrightarrow \quad Y = \frac{r \cdot d}{f} \\ Y &= IFOV \cdot r \quad \dashrightarrow \quad IFOV = \frac{Y}{r} \\ SNR &= \frac{N_p \cdot QE}{\sqrt{N_r^2 + N_p \cdot QE}} \quad \dashrightarrow \quad N_p = \frac{SNR \cdot (SNR + \sqrt{SNR^2 + 4 \cdot N_r^2})}{2 \cdot QE} \\ N_p &= \frac{H_i \cdot \lambda}{h \cdot c} \quad \dashrightarrow \quad H_i = \frac{N_p \cdot h \cdot c}{\lambda} \\ H_i &= P_{in} \cdot \tau_0 \cdot T_i \quad \dashrightarrow \quad T_i = \frac{H_i}{P_{in} \cdot \tau_0}, \end{aligned}$$

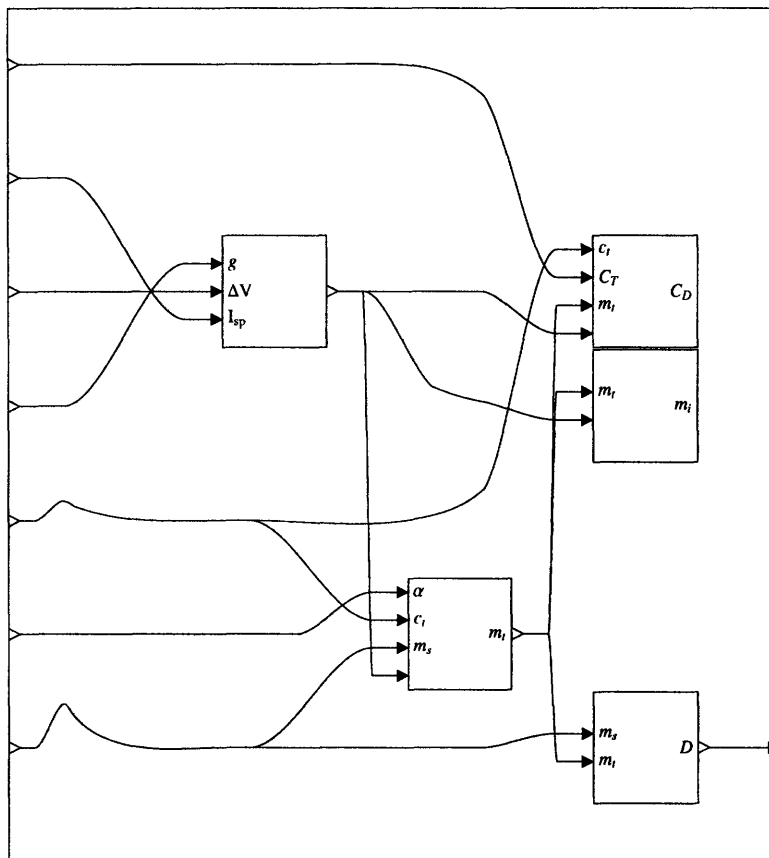


Figure 5-12: Submodel replacing that in Figure 5-11 after partial inversion

as indicated in Figure 5-13. The model obtained has SNR and C_T as input parameters and T_i as an output parameter.

Our main purpose is not to compare the plotting facilities of the two environments, but we briefly describe the plotting process in each environment. In the MDPL environment, the resulting MDPL model above is passed to a subnode with the plotting primitive discussed in §5.1 as its applicand, along with numerical primitives specifying the range of the plot. In the spreadsheet environment, the engineer must substitute values into the appropriately modified model and accumulate these values to assemble a data table from which to generate the plot. However, this does not take much time, and hence does not distort the relevance of the experiment's results to its main purpose.

In Part II.2, the model must be modified so that m_i and T_i are input parameters while C_T and λ are output parameters. In the MDPL environment, the engineer again applies the partial inversion transformation, using a permutation that swaps the new input parameters with the new output parameters. The engineer in the spreadsheet environment now faces different issues in altering the model.

Figure 5-14 shows the block triangular decomposition for this case, which contains one component involving five equations and five variables. The submodel of the new subnode replacing this set of subnodes in the final model must have output parameters L , λ , H_i , H_λ , and P_{in} and input parameters in terms of the parameters c , D , h , k , r , R , T , $\Delta\lambda$, N_p , T_i , τ_0 , and τ_λ . In this case, the equations cannot be solved for the desired output parameters in terms of elementary functions: eliminating the variables L , H_i , H_λ , and P_{in} , we find the system of equations reduces to the transcendental equation $8 \left(1 - e^{\frac{ch}{kT\lambda}}\right) r^2 N_p \lambda^4 + \pi^2 c D^2 R^2 \Delta\lambda T_i \tau_0 \tau_\lambda = 0$ in the parameter λ .

As before, purely numeric methods cannot reliably converge to the desired solution: in fact, this particular equation is quite pathological over the ranges of parameter values in the problem, and standard numerical methods may not yield any accurate solutions without a well-selected candidate solution to start from. Hence the partial inversion transformation applies a seminumeric method using the initial values already in the model from the previous step: it replaces the subnodes corresponding to the coupled equations above by a new subnode with a new primitive applicand dynamically implementing the following approach.

Consider the case for an IR telescope at ESL_2 (in this case, the cost scale factor $\alpha = 1\,000\,000$ $\$/m^3$, which is the value used in Part II.2). Figure 5-15 shows a plot of the left-hand side of the above equation as a function of λ , using the initial values remaining from Part II.1, that is, all parameters except $IFOV$ and λ have the values given in Table 5.4, while SNR is fixed at 10. The equation has two roots even over positive values of λ , so merely constraining the domain of λ to "reasonable" values is not sufficient to identify the desired solution. It can be identified by the environment only because it is already known from the initialization left from Part II.1.

In Part II.2, we fix the value of m_i while freeing λ . Consider the surface shown in Figure 5-16:

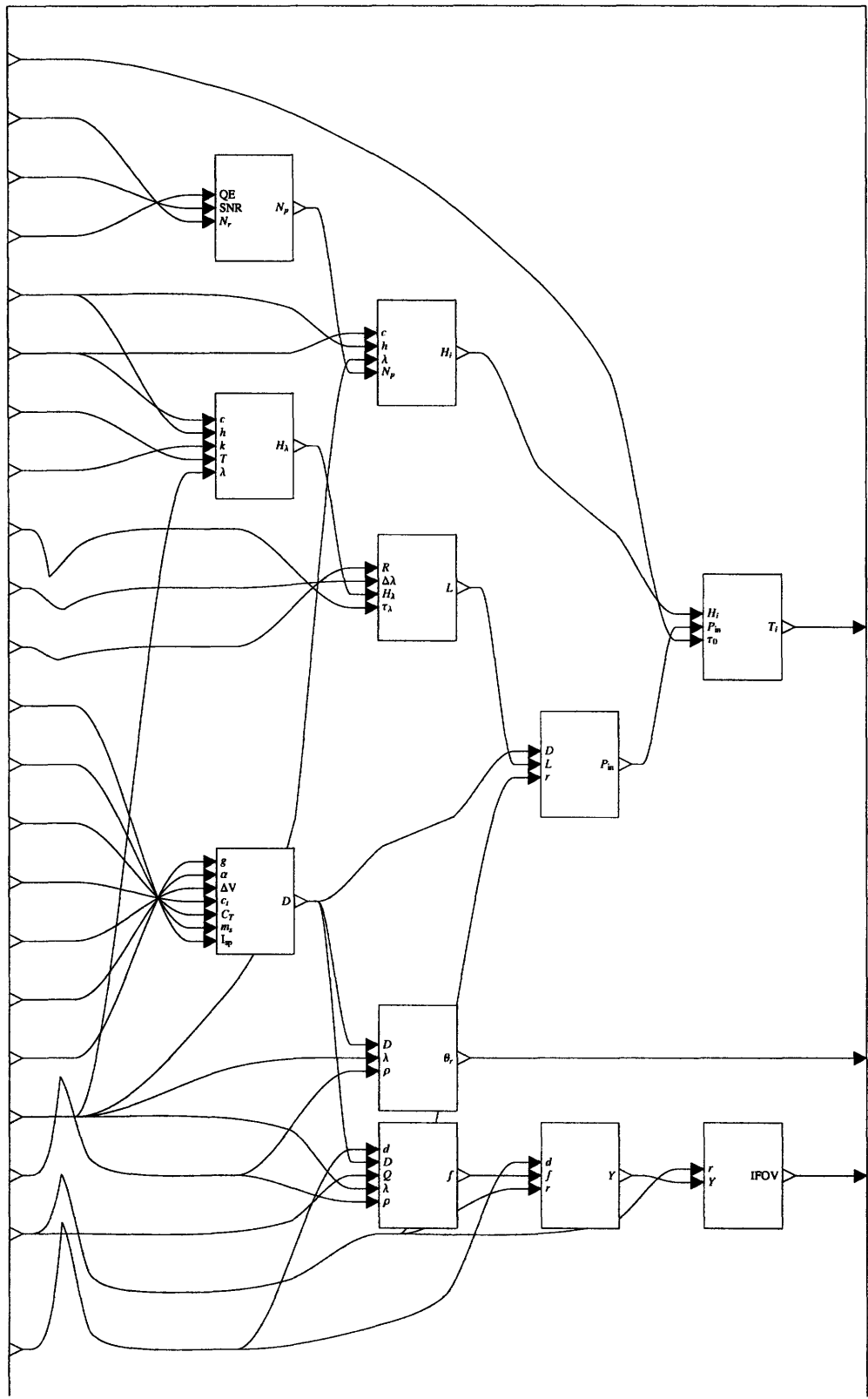


Figure 5-13: Partial inversion of the model in Figure 5-6 by a permutation swapping $IFOV$ and T_i for SNR and C_T

	c	d	m_s	g	ΔV	m_f	l_{sp}	m_t	D	Q	h	R	$\Delta \lambda$	k	T	ϵ_λ	r	T_f	t_0	QE	SNR	N_i	N_p	L	λ	H_i	H_s	P_{in}	ρ	f	Y	IFOV	α	c_t	C_D	C_T	O_x		
c	■																																						
d		■																																					
m_s			■																																				
g				■																																			
ΔV					■																																		
m_f						■																																	
l_{sp}							■																																
m_t								■																															
D									■																														
Q										■																													
h											■																												
R												■																											
$\Delta \lambda$													■																										
k														■																									
T															■																								
ϵ_λ																■																							
r																	■																						
T_f																		■																					
t_0																			■																				
QE																				■																			
SNR																					■																		
N_i																						■																	
$SNR = \frac{OE N_i}{\sqrt{N_i \cdot OE N_i}}$																						■	■	■	■														
$L = \frac{1}{4} R^2 \Delta \lambda H_i t_\lambda$													■	■											■			■											
$N_p = \frac{\lambda H_s}{c h}$	■																							■			■	■											
$H_i = P_{in} T_f t_0$																											■			■									
$H_s = \frac{2 \epsilon^2 h \lambda}{1 + \epsilon^{2 \lambda} \lambda^2}$	■													■	■													■											
$P_{in} = \frac{\rho \lambda h}{4 t_f}$																									■														
ρ																																							
$D = \frac{2 \epsilon \lambda h}{d}$		■								■	■																												
$f = \frac{d \lambda}{Y}$			■																																				
$Y = IFOV r$																																							
α																																					■		
c_t																																						■	
$C_D = D^2 \alpha$																																					■		
$C_T = C_D + c_t m_f$																																					■	■	
$O_x = \frac{\lambda h}{D}$																																						■	

Figure 5-14: Block triangular decomposition of the binary matrix relating parameters to values and equations in Part II.2

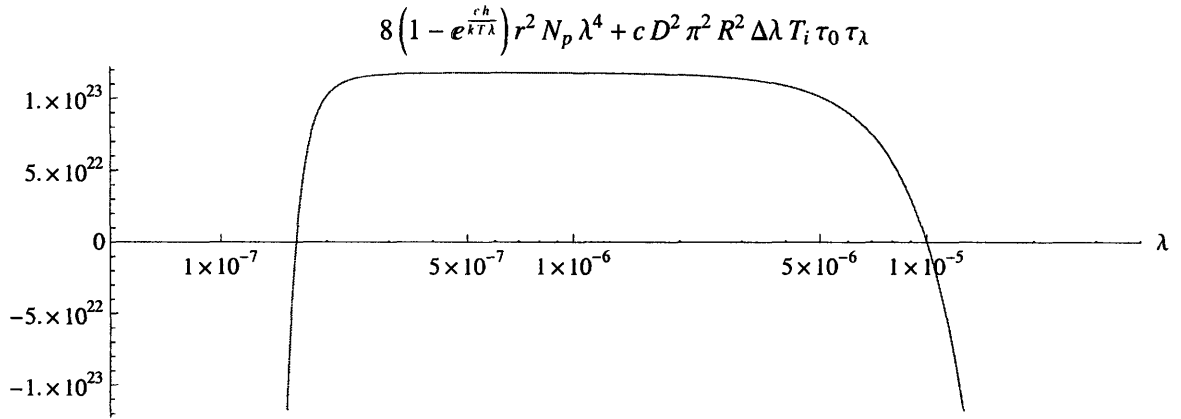


Figure 5-15: Log-linear plot of $8 \left(1 - e^{\frac{ch}{kT\lambda}}\right) r^2 N_p \lambda^4 + \pi^2 c D^2 R^2 \Delta \lambda T_i \tau_0 \tau_\lambda$ versus λ with initial values from the model in Part II.1: the rightmost root $\lambda = 1.0 \times 10^{-5}$ is the desired one

it shows how the plot of Figure 5-15 varies as m_i varies from its initial value of 39 174 kg to its new value of 60 000 kg, and how the desired root of λ moves with it. The primitive generated by the partial inversion transformation employs an iterative refinement method using tolerances based on bounds of numerical derivatives of the surface moving outward from the initial solution, to effectively trace the root locus of λ indicated by the black arrow in Figure 5-16 and converge on the correct root of λ for the new value of m_i . Later, when the value of T_i is varied to generate the plots in Part II.2, it tracks a similar root locus through variations in T_i . As with any seminumeric method, this approach can fail for sufficiently pathological functions; however, the functions encountered in the case study provide considerable testament to its robustness.

Part III: Model Extension In Part III.1, the engineer must apply the existing telescope model from Part I to two different sets of values for the input parameters, then combine these results, along with the new input parameter l , to create a model of a two-telescope system. Because both telescopes operate at the RF wavelength, and one of the two telescopes is always Earth-based, all the input parameters in Table 5.4 have the same values for both telescopes except for ΔV and α . Furthermore, the new calculations pertaining to the two-telescope system depend only on the input parameters ρ , λ , and N_r .

Taking advantage of these simplifications, the engineer using MDPL can first create a model encompassing all elements of the original telescope model except SNR along with all the input parameters except for ΔV , α , ρ , λ , and N_r , and pass the values of ρ , λ (common to both telescopes) to an instance of this model. S/he can then apply this model to two new subnodes, passing the corresponding values of ΔV and α for each of the two telescopes to one of the two subnodes. Finally, s/he can combine the results from these two subnodes using the existing submodel from Part I implementing the SNR equation along with some primitive mathematical operators, as shown in

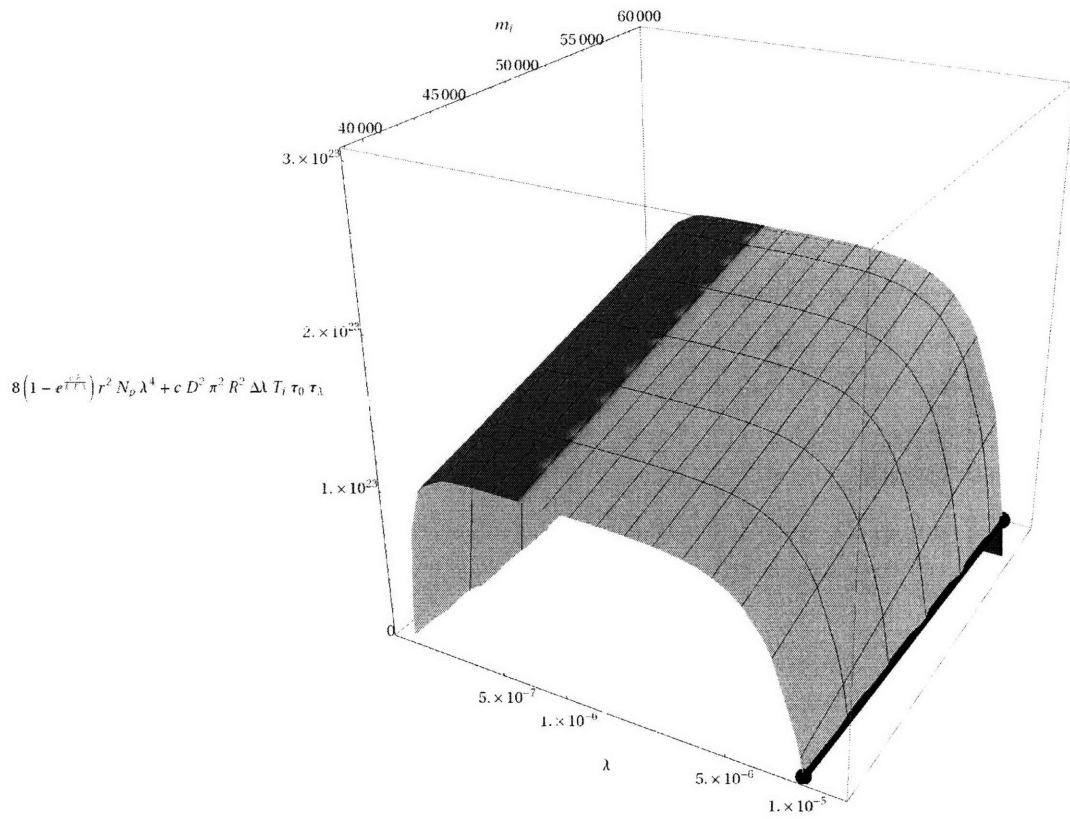


Figure 5-16: Log-linear-linear plot of $8\left(1 - e^{-\frac{ch}{kT\lambda}}\right)r^2 N_p \lambda^4 + \pi^2 c D^2 R^2 \Delta\lambda T_i \tau_0 \tau_\lambda$ versus λ and m_i with initial values from the model in Part II.1: the black arrow shows the movement of the desired root of λ as m_i moves from 39 174 kg to 60 000 kg

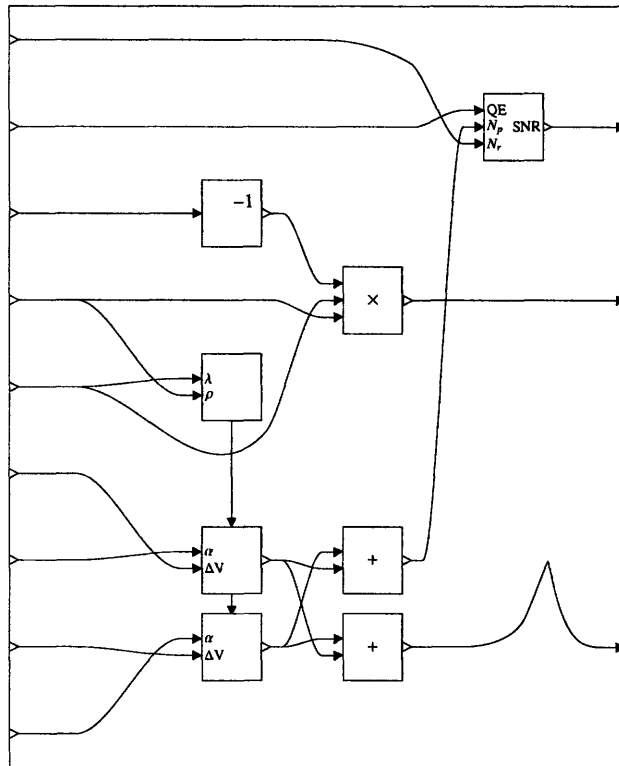


Figure 5-17: Model in Part III applying the original telescope model in two places to model a two-telescope system; note that despite the layout of the diagram, both subnodes on the bottom left are applicands of the same subnode above

Figure 5-17. This is only one possible way of structuring the two-telescope system, but it represents one with very little incremental effort.

In the spreadsheet environment, the only straightforward approach is to make multiple copies of the entire model from Part I in a new area of the spreadsheet, then fill out different copies with the different possible input parameters, and add separate copies of the new calculations for each of the two-telescope options. The simplification of the input parameters in Part III makes this approach simple enough to be feasible, but it causes even greater difficulties in Part III.2 than those faced in Part II, since all copies of each calculation must be separately replaced to reorder the dependencies among parameters.

In contrast, the MDPL implementation of Part III.2 proceeds using the same methods as in Part II: here the new variable to be solved for (D_2) must be moved through multiple nested layers to make it an input of the top-level model in order to perform the desired partial inversion transformation.

However, the seamless operation of the RLS transformation across multiple layers of the submodel hierarchy makes this no more difficult than the model transformations performed in Part II.

5.2.3 Experimental Results

The challenges of the design task and the advantages of the MDPL environment discussed in the previous section were clearly visible in the results of the experiment, though secondary differences also played a significant role in task performance. Of the five participants using the spreadsheet environment, only one produced substantive results in Part II, which he did by devoting additional time to the task beyond the 90-minute period allocated for the experiment, and none had time to begin Part III. In contrast, all but one participant using the MDPL environment produced substantive results in all parts of the design task. We now discuss the results of each part of the experiment in detail.

Part I: Model Construction All participants completed Part I of the design task. This part required roughly 45 minutes for users of both environments, though one user of the spreadsheet environment took over an hour to complete it. Participants encountered different challenges along the way that occupied their time.

Users of the spreadsheet environment spent some time organizing the parameters in the model into the tabular layout of the spreadsheet, including labeling the parameters and the values of the location and wavelength variables corresponding to different cells. Most chose to essentially copy the layout of Table 5.4, arranging values with corresponding dependencies on location or wavelength along corresponding axes of the cell table. One participant did not follow this pattern, and as a result was forced to make more manual edits to the formulas in later calculations rather than taking full advantage of the '\$' operator.

The main challenge for spreadsheet users in Part I was to write the formulas computing parameters in Table 5.6 with the correct references to other parameters and correct dependencies on the layout of parameters dependent on location or wavelength. The formula entry process was time-consuming and error-prone, since participants had to visually locate constituent parameters in their existing cell table, enter location references to these parameters, and manually edit these references to exhibit the correct dependency on location and wavelength variables (typically, adding '\$' to one or both axes of the reference before copying). All errors observed in the Part I spreadsheet calculations involved mistakes in entering formulas, and almost all of these involved incorrect references. As a result of such errors, only one participant using the spreadsheet environment obtained entirely correct results in Part I, while others had error rates of 20-90%. Multiple participants noted that it was very difficult to catch errors in entering formulas.

Users of the MDPL environment had fewer difficulties with formulas, since they could construct

the entire single-telescope model simply by entering in the equations as they appeared on the instruction sheet. The correctness of references in this model was fairly easy to verify, since subnodes were automatically labeled with the corresponding variable names and the connections explicitly shown. The formulas themselves were hidden from the top-level view in this prototype environment, but could be quickly and intuitively inspected for those who chose to do so. The only error in the Part I results among MDPL users derived from one participant who entered an equation with incorrect grouping of parentheses. (This type of syntax error might not occur at all with a fluid interface for direct entry of formulas as MDPL models.)

The main challenge for MDPL users in Part I was to enter and link given values for the parameters in Table 5.4 to the model using the graphical user interface of the prototype environment. Rather than simply copying the table of values as written on the instruction sheet, entering each parameter required creating a numeric primitive, entering the number in the field, and then linking the primitive to the corresponding lists. This task was still feasible due to the relatively small number of distinct parameter values used in the model, but clearly larger modeling tasks could benefit from improved mechanisms for data entry, particularly of tabular data.

MDPL users took some time figuring out how to assemble the parameters of Table 5.4 into lists and apply the telescope model to them all together, as in the construction of Figure 5-9. Most initially inquired about a way to “copy” the model, as one would approach the problem in a spreadsheet environment. Each eventually developed a slightly different method to apply the model across all locations and wavelengths without copying, illustrating the flexibility of the MDPL language.

Overall, the advantages of the spreadsheet environment in rapid data entry and the advantages of the MDPL environment in rapid formula entry resulted in a wash in terms of task completion time. However, the automation of variable references in the MDPL environment appears to have contributed to reduced error rates in model construction.

Part II: Trade Analysis All participants had sufficient time to comprehend the challenges of Part II. The users of the MDPL environment typically completed Part II in about 20 minutes (though one participant required nearly 40 minutes), and all produced results of the intended form. In contrast, none of the users of the spreadsheet environment obtained results for either task in Part II within the allotted time period: their accomplishments varied from no observable progress to the solution of some of the less challenging equations to be solved in Part II.1. One participant using the spreadsheet environment devoted an additional 37 minutes beyond the experiment period to produce some plots providing the essential information requested in Part II.1, though in a different form.

The main challenge for MDPL users in Part II was to insert new primitives for generating plots

and supplying values for the new input parameters, and connect these primitives to the appropriate places in the model after applying the partial inversion transformation to the original submodel describing the telescope. All users succeeded in incorporating the plotting function into the model containing the telescope submodel, which was then mapped over the location and wavelength variables without any further modifications, producing a normal form consisting of a matrix of plots corresponding to the different location and wavelength values. As in Part I, most of users' time was spent creating and linking new numerical values in the model. Most suggestions addressed the difficulties of the user interface: multiple users complained about the messy visual layout of the larger models and the need for a great deal of scrolling when they could not fit entirely on one screen.

Most users of the spreadsheet environment succeeded in determining the variables for which each equation should be solved to complete Part II.1. One participant even drew directed graphs on paper showing the modified relationships between variables through equations based on the desired permutation of inputs and outputs (Figure 5-18), conceptually similar to the graphical representation of the corresponding MDPL model after partial inversion. Though entirely unaware of the MDPL language and representation, he speculated that a "tool with small graphs showing the input and output for each equation would be useful."

Most participants succeeded in solving the quadratic equation $SNR = \frac{N_p \cdot QE}{\sqrt{N_p^2 + N_p \cdot QE}}$ by hand for $N_p = \frac{SNR \cdot (SNR + \sqrt{SNR^2 + 4 \cdot N_p^2})}{2 \cdot QE}$ (though at least two made mistakes in solving it the first time), along with several of the simpler equations. One attempted to combine multiple simpler equations to eliminate variables, though any success using this approach was not apparent. However, no participant made substantive progress in solving the cubic equations for C_D , D , m_i , and m_t discussed above. At least two tried using the numerical solver in Excel[®] but did not succeed in obtaining usable results, even when used incrementally on single equations. When asked what facilities would improve their capability to execute the design task, multiple participants suggested tools that would "backtrace the equations for you" like the partial inversion transformation in MDPL.

The participant in the spreadsheet group who produced plots in Part II.1 after additional time did so by plotting total cost C_T for particular values of T_i rather than the other way around. He then manually adjusted the range of T_i values to correspond to the requested range of C_T values in the task description. This produced a plot expressing the same relationship between variables as the one requested (Figure 5-19), and avoided the need to solve cubic equations to express other variables in terms of C_T . This approach produced an effective result in this particular case, but becomes less likely to succeed as models grow in their number of parameters and equations.

Altogether, participants using the spreadsheet environment faced challenges in identifying the modifications necessary to express the model in terms of new input variables, and encountered insurmountable difficulties in computing these modifications either numerically or algebraically by hand. As one participant put it, "I think that if I had know[n] task number two beforehand I [would

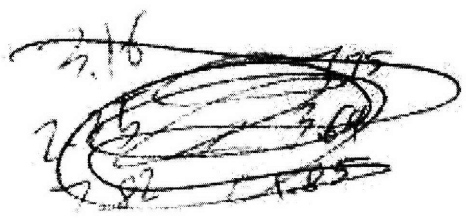
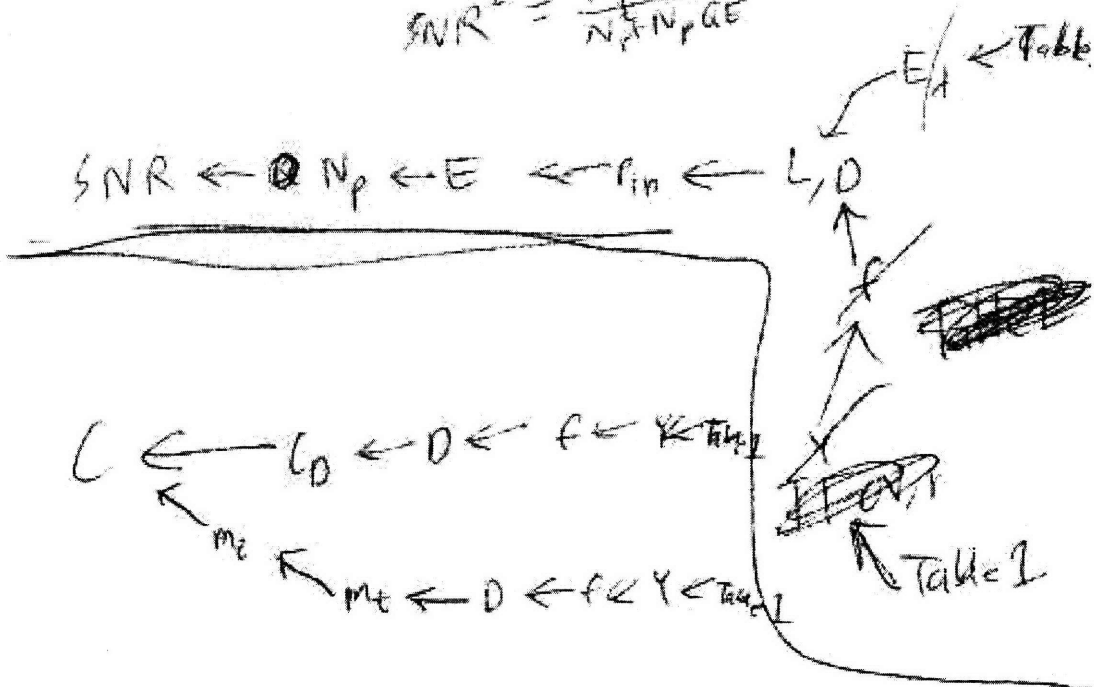
Want C, θ_r, SNR

~~Pin~~ $P_{in} = \frac{E}{L \cdot D}$

$$E = \frac{N_p h c}{\lambda}$$

$$\frac{(\text{SNR})^2}{GE} (N_s + N_p GE) = \frac{P_r}{P_t}$$

$$\text{SNR}^2 = \frac{N_p^2 GE^2}{N_s + N_p GE}$$



$$\theta_r \leftarrow D \leftarrow f \leftarrow Y \leftarrow \text{Table 1}$$

Figure 5-18: Notes made by one participant using the spreadsheet environment in Part II.1, including diagrams similar to those automatically generated by the graphical representation of models in the MDPL environment

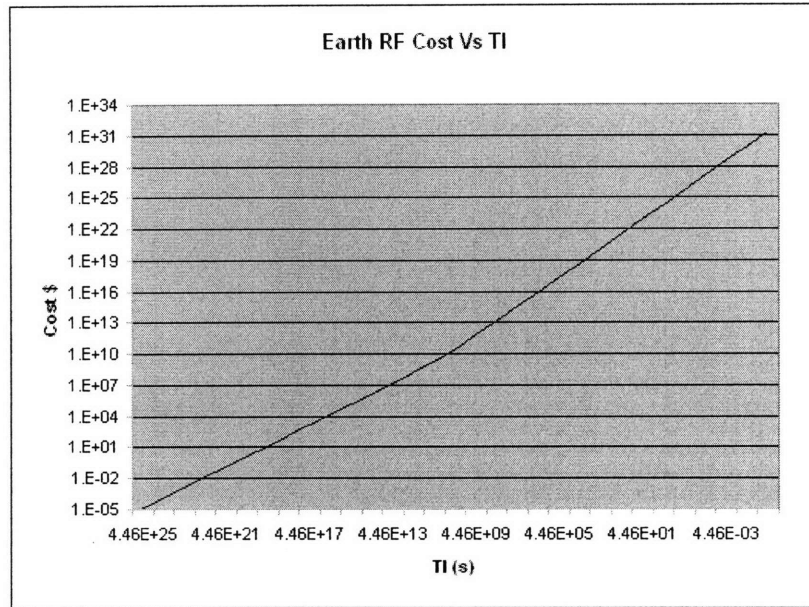


Figure 5-19: Log-log plot of Earth-based RF telescope total cost (C_T) versus integration time (T_i) by the only participant using the spreadsheet environment to produce plots in Part II.1

have set up] the problem in a different manner. However, [I would have] been facing the same difficulty if the parameters to study would change.”

It is possible that the brief introduction to the features of the MDPL environment helped to guide these participants thinking in a way that aided their completion of the design task. However, while users of the MDPL environment were provided reminders upon request about the functionality of various built-in features of the environment, they were not given any guidance about how to proceed in solving the task. Moreover, the additional features built in to the environment over and above the language itself merely provided conceptually similar functionality to the Excel[®] environment for interpreting mathematical syntax and assembling lists or tables of data. Hence we attribute the observed differences in performance on Part II mainly to the inherent facilities of the MDPL language.

Part III: Model Extension No participants using the spreadsheet environment had time to begin Part III within the experiment period. All but one of those using the MDPL environment produced results for Part III, these taking roughly 30 minutes to do so; the remaining participant had spent nearly 40 minutes in Part II, and did not have time to make substantial progress on Part III. Some participants went over the allotted 90-minute period by a few minutes in order to complete the task, while others stopped after completing the initial model in Part III.1. These participants felt that Part III.2 would be straightforward to complete using the approach they had been through already in Part II.

Part III required more time than Part II, mainly due to the mechanical challenges of constructing the two-telescope model. It took a few minutes for participants to realize which intermediate results were dependent on which variables, and how to make most efficient use of the existing telescope model. However, most of the time was spent restructuring the model and adding primitives and connections for the new calculations.

The main challenges again pertained largely to user interface issues. Participants felt it cumbersome to perform model transformations by selecting elements in one model and then executing actions after moving up or down the submodel hierarchy to another model, at which time the original selection was no longer visible. Multiple participants suggested a continuously zoomable interface in which the visibility of elements would be limited only by their size, and in which elements could be dragged from one submodel to another even across multiple levels in the hierarchy. This approach was considered in the development of the prototype environment, but was abandoned due to the limited time available for software development.

Overall, participants were able to make use of the MDPL model transformations as they were intended to be used, resulting in a substantially cleaner approach to Part III than would be possible in the spreadsheet environment. Some felt that symbolic programming languages like MATLAB would be preferable for making these types of modifications to a mathematical model, but agreed that the clear layout provided by the visual representation in MDPL helped them to structure their approach to the tasks in Part III. Besides the automation of structural changes to programs, this may well be one of the principal advantages of the MDPL language.

5.3 General Features of the MDPL Language

The case study in the previous section provided some insight into how MDPL improves users' ability to develop and modify programs to address engineering design tasks. Here we consider the broader advantages of MDPL for general-purpose computer programming. While the case study compared the MDPL language environment with that of the common spreadsheet, here we consider comparisons with several other programming paradigms. In a number of areas, we will describe how MDPL captures, and in some cases supersedes, well-known features of other families of languages.

The introduction in §1.2 already discussed how MDPL combines the strengths of procedural and functional language approaches, providing the flexibility of the former with the structural properties of the latter. Under each of the following headings, we describe a capability provided by explicit features of existing languages and show how it arises naturally, sometimes in a more general form, in MDPL. These capabilities cover a large subset of the important features emphasized in modern programming languages.

Reference Calls Most languages provide mechanisms for referring to values elsewhere in a program without copying them: this ensures that changes are propagated to other parts of the program that make use of the values, and avoids the inefficiency of duplicating them. This functionality is commonly implemented as “call by reference” or “pointers” in procedural languages, and by maintaining a symbol table in functional languages. MDPL provides this functionality by displaying intuitively in the visual representation the flow of values through the program, and by providing an instantaneous mechanism (through model transformations) to pass any value to any valid location in the program, including values that represent entire models. At the same time, an efficient implementation of MDPL (§6.2) will automatically avoid the inefficiencies of copying where possible by analyzing the link graphs of the MDPL models it evaluates.

Closures Several functional programming languages provide a mechanism to define an inner function within an outer function so that the values of the outer function’s arguments when it is executed are preserved in the definition of the inner function, even when the inner function is passed elsewhere in the program. This avoids the need in symbolic functional languages to explicitly pass all relevant parameters of the outer function as arguments to the inner function, and avoids some of the difficulties in modifying functional structure that we have described earlier. In MDPL, with no symbolic variables, an explicit feature analogous to closures is unnecessary: the desired behavior occurs naturally when interior models receiving parameters from their parent models are passed elsewhere, and with essentially no overhead to modifying argument structures of models (thanks to model transformations), parameters are simply passed explicitly wherever they are needed.

Macros In the sense of functional programming languages, macros essentially provide a mechanism to define functions that control the evaluation of their arguments so that, for example, arguments that are not well-defined are not evaluated. We have seen an example of this approach in MDPL with the “Defined?” model in Table 5.2. Such models are generally implemented based on primitives with special evaluation rules: these can be extended with controlled evaluation by “wrapping” expressions not to be evaluated inside new submodels, passing those submodels as values, and applying them to other subnodes to extract their contents when desired. With a non-symbolic syntax, MDPL also helps prevent common pitfalls of macros in symbolic languages like multiple evaluation and variable name clashes.

Function-Level Programming As described briefly in the introduction, function-level programming is an approach emphasizing the construction of functions by applying function-level operators to existing functions, without explicitly describing operations performed on primitive values. This approach encourages programmers to build a more generally applicable ‘algebra’ of functions within their programs that can be used to quickly generate new functionality. MDPL supports this approach

through its ability to build models that operate on other models; moreover, it allows a great deal of flexibility in building function algebras by permitting multiple inputs and outputs of functions to be connected in any possible way.

Symbolic Programming Languages like LISP are often used for symbolic programming, an approach based on building up large symbolic expressions and applying transformations to them with a program. This approach is particularly important in advanced applications like computer algebra and language processing, and can also facilitate efficient implementation of recursive functions by limiting the stack depth of recursive calls and enabling memoization. It is made possible by the ability to build up large structures in a language and treat them as values in the same manner as primitive objects. This is naturally possible in MDPL, since one can construct large models that are normal forms (hence have their structure preserved under the normalization transformations) and can easily manipulate them like primitives. In fact, because of the more general form of MDPL expressions relative to common symbolic languages (acyclic digraphs rather than strict trees), MDPL allows expressions to be entered in a way that more easily takes advantage of their functional structure.

Object-Oriented Programming Many modern languages encourage an approach in which related data and functions are grouped together as “objects” of which multiple instances can be constructed with different parameter values, and in which new classes of objects can be defined that inherit some or all the elements of existing classes of objects. Like closures, MDPL naturally implements objects as models with multiple outputs representing the data or function elements of the object, of which instances are constructed by passing parameters to subnodes applying the defining model. Different object models can share elements in the manner of inheritance by defining an external model applicand representing the shared elements and specializing it via the currying approach described in §3.2.

Introspection It is often important for a program to be able to monitor and analyze its own internal behavior, whether for user-level debugging or some form of algorithmic feedback. Procedural languages often fulfill this function using global variables that span an entire program, and can easily become confusing to manage; purely functional languages typically make this cumbersome, since accessing a value deeply nested within the function hierarchy requires rewriting the argument structures of the surrounding functions, and often paying an efficiency penalty to create more copies of the values in question. The model transformations of MDPL make it trivial to access values from deep within a model, while automatically preserving the functional structure of the model and encoding this structure in a way that a compiler can easily take advantage of to avoid needless copying.

Sequencing Procedural languages natively operate in terms of sequenced operations, but functional languages (based on evaluation of expressions) do not provide an explicit mechanism for specifying the sequencing of operations. Control of operations sequencing can be important for features like input/output with external users or programs, or keeping track of changes in state variables. Implementing these in a rudimentary functional language requires passing the relevant values in and out of every function in which they are modified (sometimes through many nested layers of surrounding functions) to control the order in which these modifications take place. Some modern functional languages have implemented advanced features like monads to help facilitate this process without requiring such extensive modification of syntax; however, because of inherent limitations of the syntax of symbolic functional languages, monads are tricky and error-prone to use. MDPL eases the burden of passing values throughout the function structure of a program, obviating entirely the need for monads or other workarounds for sequencing operations.

The above examples indicate how the generic power and flexibility of the MDPL language provide a wide range of capabilities that other languages address with specialized features or design choices. Certainly there are syntactic features, such as list comprehensions paralleling mathematical set definitions, that cannot be directly realized in MDPL. However, the above examples capture a large portion of the fundamental features that largely determine the ease of writing programs in a particular language.

Chapter 6

Future Research & Development

This work has emphasized the theoretical foundations of the MDPL formalism and described some of the practical features of the resulting language, particularly with respect to engineering design analysis. However, a great many issues remain to be resolved to implement these ideas in robust engineering tools or, ultimately, general-purpose programming platforms. Some of these issues are the subject of active research in other domains, and can take advantage of ongoing work in these areas. In addition, the core ideas of MDPL suggest possible extensions to other application areas not yet explored.

6.1 User Interface Development

The experience of users in the case study of S5.2 has indicated the importance of user interface design for a practical MDPL implementation. Some of the opportunities for improvement in this area include:

Automated Visual Layout of MDPL Models This problem addresses many of the same challenges facing layout algorithms for general digraphs and acyclic digraphs, but differs in certain key respects. Unlike ordinary acyclic digraphs, MDPL diagrams intuitively contain multiple factors of directional “flow,” one dealing with passing of inputs and outputs, the other with application of nodes to one another; of course, these two can be mixed (source applicands or node arguments), further complicating the realization of intuitive layout. MDPL diagrams are also challenging in that subnodes should ultimately have different sizes reflecting their contents or labels and their argument structures. Additionally, the local structure of arrows around subnodes matters a great deal to unambiguous, aesthetic layout, so the layout algorithm must combine both global and local routing constraints, as the prototype environment attempts to do. Ideally, a layout algorithm would recognize certain types of symmetry in the structure of a model and reflect these symmetries in the

visual layout. Finally, the hierarchical nesting of models and the connections of sources/sinks and applicands across their boundaries imposes additional layout constraints within which to optimize the layout of individual models. Advanced layout algorithms for MDPL will probably adopt a recursive decomposition approach like those used for large graph layouts, but the structure of the constraint space for MDPL models is much more complicated, and will likely require strong heuristic methods imitating the behavior of human drawing methods.

Intuitive Mechanisms for Navigating Models Perhaps even more than visual layout, users of the prototype environment wished for better ways to navigate MDPL models. Improved layout will help to fit more information on a single screen, reducing the need for scrolling, but navigating selections is still important for facilitating rapid entry of model data. Participants in the case study suggested a zoomable view capable of showing multiple levels of the submodel hierarchy at once, which seems a reasonable approach: indeed, this was the original intention of the visual representation as specified in §3.2. Such a view would require hiding elements of submodels when they reached a certain size threshold, and making them reappear once large enough to manipulate. Keyboard shortcuts could also be useful in navigating models: one can imagine quickly moving among the nodes, sources, and sinks of a model using arrow keys to make selections and even enter primitives by name, rather than the mouse-intensive GUI of the prototype environment.

Auxiliary Information Attached to Models Much like integrated development environments (IDEs) for symbolic languages that augment the sparse code of the language with a richer environment for the programmer, MDPL models could benefit from a number of non-essential enhancements to their basic encoding. The prototype environment already provides labels for sources and sinks based on the variable names in equations automatically translated into MDPL models. Future environments could give users control over labels on sources and sinks, and even on subnodes and arrows. The environment could even infer labels automatically from the labels of model elements connected to them by the user. Combined with the intuitive visual layout of models, such labels could serve as extremely useful documentation features. To aid programming, sinks of standard MDPL models known to the development environment could be enhanced with dropdown menus providing common choices of inputs for the corresponding arguments. For models with many possible inputs or outputs, some sinks or sources could be hidden by default and made visible by the user as necessary; some of these could have initial values created along with them when a standard model is instantiated, providing “default” values for “optional” arguments like some symbolic languages.

Collaborative Environments for Team-Based Programming The formalism and visual representation of MDPL together offer a unique opportunity for fluid automated management of collaborative programming. Collaborative software development in symbolic languages generally involves

file-by-file management of large blocks of code, with one user restricted to edit one file at a time. This often results in resource conflicts, and requires frequent reorganizations of code to avoid individual files becoming large enough to inhibit collaboration or small enough to inhibit user development. The hierarchically nested structure of MDPL models suggests a different approach, based on dynamic resource locking of submodels according to a user's current work region in a model. In this approach, an entire team of people could edit the same model at the same time, provided that (for instance) no two people made edits to the same submodel at the same time; the assignment of users to submodels would be determined by the current view within each user's interface window and a priority system for allocating ownership. In such a system, users could still observe other users' changes to any part of the model in real time, greatly enhancing the speed of communication in software through the project.

6.2 Efficient Implementation

Considering the strong relationship between the MDPL formalism and that of combinatory logic, it is logical to consider compilation strategies based on reduction to simple combinator calculi. This type of strategy has been successfully developed for modern functional languages, and has the advantage that combinator reduction rules can be implemented substantially faster in machine code than the symbolic transformations analogous to β -reduction in the lambda calculus [70, 69].

Efficient evaluation of MDPL models faces many of the same challenges as those faced by symbolic functional languages. The normalization transformations of MDPL, like normal order reduction in lambda calculus, naturally lead to so-called "lazy evaluation," which creates copying inefficiencies if applied in a brute-force approach. The solution developed for this problem in symbolic functional languages is "normal graph reduction," in which the copying of terms is imitated by the interpreter using acyclic digraphs, with multiple pointers to single addresses in memory [74]. This approach is natural for MDPL, whose native representation already follows this form. This approach can also support an efficient form of recursion by introducing cycles in the graph where fixpoint models are applied, as implemented in languages like SASL and Miranda [68, 71].

For implementation on machine hardware, it is useful to tailor the choice of low-level combinators and associated reduction rules to correspond to the functions appearing in the program being compiled. This idea has been extensively developed for symbolic languages based on lambda calculus in the form of " λ -lifting" [37]. A similar mapping from MDPL graph reductions to a conventional hardware architecture may enable highly efficient compilation of MDPL models.

The use of MDPL for time-critical engineering design tasks will necessitate efforts toward optimizations beyond those of traditional compilation that can take full advantage of the computing resources available to design teams. Automated decomposition of MDPL graph reductions using

existing algorithms can provide opportunities for parallelization across many processors, which can be particularly advantageous in performing trade studies over a space of design parameter values. In an interactive environment, it may also be possible to perform intelligent automated memoization of key intermediate results (for instance, based on typical computation times) to maximize the use of memory resources available. Additional optimizations may apply to specific types of tasks, and will most likely parallel many of the developments for symbolic languages.

6.3 Algebraic Transformations

We have given a high-level description of the implementation of the partial inversion transformation through the description in §4.3 and the demonstration in §5.2.2. However, the details of the implementation touch on many areas of active research in symbolic and numeric computation, and will likely evolve as research in these areas progresses. The prototype environment of §5.1 makes use of many internal algorithms of the *Mathematica* platform, and thus we cannot even give a complete characterization of the methods used here. However, some recent developments will certainly influence the efficiency and generality achievable in the partial inversion transformation.

In the case of polynomial and related systems of equations, methods for efficiently computing Gröbner bases continue to advance. Of particular interest are “Gröbner walk” methods that can be used to generate Gröbner bases with respect to the lexicographic monomial orderings useful in equation solving from more easily computed bases with respect to alternative monomial orderings [13]. For certain monomial orderings, research has made steady improvements in heuristics for variable ordering, which can often take advantage of specialized structure in the problems to be solved [17]. In addition, numerical implementations of Gröbner basis methods based on significance arithmetic have substantially increased the size of polynomial systems for which Gröbner bases can be feasibly computed [76].

In the case of equations without algebraic solutions, the efficient implementation of seminumerical substitutes based on initial solutions remains a key challenge. There is a natural trade-off between efficiency and robustness influenced by the management of error tolerances in selecting from among multiple solutions, and a good deal of application-specific calibration may be necessary to achieve the right balance. For particularly difficult cases, it may be useful to pursue an approach based on polynomial approximation (e.g., Padé approximations) combined with robust numerical solvers for polynomial systems [26, 35]. However, engineers’ intuition typically tends to focus the design tradespace on areas where the functions relating design variables are relatively well-behaved.

6.4 Core Language Extensions

Considerable effort has gone into developing a tight yet expressive formalism for the MDPL language. However, certain augmentations of this formalism may enable the main ideas of the MDPL language to be applied to more specialized areas with additional requirements. Here we discuss a few ideas for extensions of the core language with their associated interpretations and applications:

Dynamic MDPL In real-time systems, changes in input values often change more rapidly than the desired results of these changes can be fully evaluated. This occurs in applications as mundane as user interfaces and as critical as flight control software. When updates must be managed according to available processing power, one often wishes to ensure that certain updates are completed based on information from a consistent point in time. In MDPL, it is natural that certain subnodes (for instance, those sending signals to actuators) should wish to enforce the time-consistency of their relevant inputs: hence the core language could be augmented by indicators of tolerances in the asynchronization of inputs to particular subnodes. These could be applied in a nested fashion in order to manage tolerances throughout the function hierarchy, and enforced in the code generated by a real-time compiler.

Typed MDPL Some software engineering practices encourage enforcement of type consistency to reduce the potential for catastrophic errors in execution. Such practices are frequently used for critical systems in many aerospace applications. To enforce a strong type system, one must either give up the goal of universal computation, or augment the language with a mechanism for explicit recursion [72]. MDPL has the latter mechanism in the form of the RNX transformation, meaning that it can easily be adapted into a universal typed language. A type system may have advantages for collaborative software development in engineering design, since it may encourage a more modular approach based on consistent mathematical structures. Such a type system could be implemented as a partially ordered hierarchy of increasingly specialized types, with functions defined in terms of minimally restricted types as inputs. This would encourage a relatively small family of functions to span common mathematical operations across a wide range of applications.

MDPL for Simulation The Simulink[®] language was mentioned in the introduction for its application of a similar formalism to the MDPL language in a different domain, that of continuous-time signal processing. MDPL could be adapted to provide an even more general language for this domain by adding a primitive to the formalism representing the application of a time derivative from input to output, and allowing cycles in the link graphs of MDPL models. In the interpretation of this formalism for simulation, the arrows of the diagram would then represent continuous functions of time rather than finite values, as in Simulink[®]. However, the resulting structure of the language

would lead to a more general domain of problems paralleling that expressed by modular modeling and simulation languages like Modelica [24]. The advantages of the MDPL representation and model transformations could prove extremely useful in this domain for modifying the physical architecture of systems being simulated. The implementation of such a language would require the solution of systems of differential-algebraic equations (DAEs) rather than simply ordinary differential equations (ODEs), and would demand the application of relatively recent algebraic methods in DAE index reduction [42]. The resulting environment could be an important counterpart to the parametric analysis tools for engineering design analysis emphasized in this work.

Chapter 7

Conclusion

This work introduces a new formal model of computation called MDPL, which forms the basis of a new kind of programming language combining the major strengths of existing programming paradigms. Foremost among these strengths are the flexibility to change relationships among elements in a program, and the power to reuse and redeploy existing structures of such relationships in new places within a program. While existing languages generally trade one of these capabilities for the other, the MDPL language combines these capabilities by using a highly structured representation augmented with algorithmic transformations on the representation. The former allows program structures to be easily identified and reused in a way that enables efficient implementation; the latter provide an immediate mechanism for making certain useful kinds of changes to a program while maintaining the constraints of the representation.

The unique combination of programming capabilities offered by the MDPL language, combined with appropriate infrastructure, allows engineers to address time-critical design analysis tasks more effectively than they are able to do with existing modeling tools. The MDPL programming environment allows engineers to quickly modify models to explore new architecture options, while at the same time quickly expressing these new options in terms of existing models. In addition, the MDPL formalism appears to offer a clear and intuitive representation of program structure that helps guide engineers' thinking about a design analysis task.

This work demonstrates that the MDPL formalism has the theoretical properties necessary to define a consistent, practical interpretation of MDPL models as computable functions. We formally define such an interpretation as a reduction algorithm applying various normalization transformations, which will reduce any normalizing MDPL model to its normal form in a finite number of steps. We also demonstrate that this interpretation makes the MDPL language universal, hence capable of implementing all computable functions. Finally, we formally define the additional transformations on MDPL models that alter the structure of these models in useful ways, and allow programmers to

build new models from existing ones.

This work also demonstrates empirically some benefits of an MDPL-based modeling environment relative to the spreadsheet-based environment that currently dominates conceptual design centers in the aerospace industry. An experimental case study compares these two approaches to modeling trade-offs in a small but real design analysis task evaluating architectures for ground- and space-based telescopes. The results of this case study indicate that while the robust deployment of MDPL in industrial-scale engineering design tools faces some challenges in user interface development, some simple approaches to these problems provide a starting point sufficient to give engineers a substantial advantage in using MDPL. In the discussion, we explain how MDPL captures many of the important features of existing programming languages within its system of model representation and model transformations.

The key technical contributions of this work are the following:

- formal definition of a new model of computation, the MDPL model, that generalizes some aspects of existing models like the lambda calculus and combinatory logic;
- theoretical justification for important practical properties of the MDPL formalism, primarily the confluence of its normalization transformations and existence of normal order reduction;
- formal definition of transformations that preserve the structural constraints and/or normal forms of MDPL models while allowing flexible changes to the relationships of elements within models;
- a structured algorithmic approach to recursive hierarchical symbolic/numeric permutation of initialized input/output variables in models representing functions with certain restrictions typically satisfied in engineering design.

In addition, this work has touched on a number of important secondary problems, including the intuitive visual representation of complex engineering models, compact representations of specialized mathematical functions, and the architecture of user interfaces for the MDPL language. Solutions to these problems will play a critical role in the future deployment of MDPL in engineering practice.

Bibliography

- [1] J.A. Aguilar and A. Dawdy. Scope vs. detail: the teams of the Concept Design Center. In *IEEE Aerospace Conference Proceedings*, volume 1, pages 465–481, 2000.
- [2] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical algebraic decomposition I: the basic algorithm. *SIAM Journal on Computing*, 13(4):865–877, 1984.
- [3] J. Backus. The Algebra of Functional Programs: Function Level Reasoning, Linear Equations, and Extended Definitions. *Proceedings of the International Colloquium on Formalization of Programming Concepts*, pages 1–43, 1981.
- [4] M. Bandecchi. The ESA Concurrent Design Facility (CDF): Concurrent Engineering applied to space missions assessment. In *NLCOSE*, 2003.
- [5] Claude Berge. Two theorems in graph theory. In *Proceedings of the National Academy of Sciences*, volume 43, pages 842–844, 1957.
- [6] Bruno Buchberger. Introduction to Grobner bases. In *Grobner bases and applications*, pages 3–31. Cambridge University Press, April 1998.
- [7] Bruno Buchberger. Groebner bases and systems theory. *Multidimensional Systems and Signal Processing*, 12:223–251, 2001.
- [8] Frank Cameron. Automatic generation of efficient routines for evaluating multivariate polynomials arising in finite element computations. *Advances in Engineering Software*, 28(4):239–245, June 1997.
- [9] Martine Ceberio and Vladik Kreinovich. Greedy algorithms for optimizing multivariate Horner schemes. *ACM SIGSAM Bulletin*, 38(1):8–15, March 2004.
- [10] Noam Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2(2):113–123, 1956.
- [11] Alonzo Church. An Unsolvable Problem of Elementary Number Theory. *Amer. J. Math.*, 58:345–363, 1936.

- [12] Thomas M. Coffee. The Future of Integrated Concurrent Engineering in Spacecraft Design. Technical report, Massachusetts Institute of Technology, 2006.
- [13] S. Collart, M. Kalkbrenner, and D. Mall. Converting bases with the Grobner walk. *Journal of Symbolic Computation*, 24:465–469, 1997.
- [14] George E. Collins, Jeremy R. Johnson, and Werner Krandick. Interval arithmetic in cylindrical algebraic decomposition. *J. Symb. Comput.*, 34(2):145–157, 2002.
- [15] Haskell B. Curry and Robert Feys. *Combinatory Logic Vol. I*. North Holland, 1958.
- [16] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, Oct 1991.
- [17] S.R. Czapor and K.O. Geddes. On implementing Buchberger’s algorithm for Grobner bases. *Proceedings of the fifth ACM symposium on Symbolic and algebraic computation*, pages 233–238, 1986.
- [18] O.J. Dahl. The roots of object orientation: the Simula language. *Software pioneers: contributions to software engineering table of contents*, pages 78–90, 2002.
- [19] O.J. Dahl and K. Nygaard. SIMULA: an ALGOL-based simulation language. *Communications of the ACM*, 9(9):671–678, 1966.
- [20] Olivier L. de Weck. Telescope Relationships, February 2007.
- [21] Johann Peter Gustav Lejeune Dirichlet. *Journal für Reine und Angewandte Mathematik*, 4:157, 1829.
- [22] A. L. Dulmage and N. S. Mendelsohn. Two algorithms for bipartite graphs. *Journal of the Society for Industrial and Applied Mathematics*, 11(1):183–194, March 1963.
- [23] Mark Edel. The Tinkertoy graphical programming environment. *IEEE Trans. Softw. Eng.*, 14(8):1110–1115, 1988.
- [24] H. Elmqvist, SE Mattsson, and M. Otter. Modelica: a language for physical system modeling, visualization and interaction. *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pages 630–639, 1999.
- [25] Wolter J. Fabrycky and Benjamin S. Blachard. *Life-Cycle Cost and Economic Analysis (Prentice Hall International Series in Industrial and Systems Engineering)*. Prentice Hall, Englewood Cliffs, NJ, 1991.

- [26] Steven Fortune. Polynomial root finding using iterated eigenvalue computation. In *ISSAC '01: Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pages 121–128, New York, NY, USA, 2001. ACM.
- [27] Abraham A. Fraenkel. Zu den Grundlagen der Cantor-Zermeloschen Mengelehre. *Mathematische Annalen*, 86:230–237, 1922.
- [28] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18(1):23–38, March 1986.
- [29] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Trans. Software Engineering*, 19(3):214–230, 1993.
- [30] E. Ghittori, M. Mosconi, and M. Porta. Designing and testing new programming constructs in a data flow VL. Technical report, Universit'a di Pavia, Pavia, Italy, 1998.
- [31] T. Green and M. Petre. Usability analysis of visual programming environments: A cognitive dimensions framework. *J. Vis. Lang. Comput.*, 7:131–174, 1996.
- [32] J.C. Heim, K.K. Parsons, S.F. Sepahban, and R.C. Evans. TRW process improvements for rapid concept designs. In *IEEE Aerospace Conference Proceedings*, 1999.
- [33] John E. Hopcroft and Richard M. Karp. An $n^5/2$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973.
- [34] Thomas Jech. *Set Theory*. Springer, third millennium edition, revised and expanded edition, 2003.
- [35] M. A. Jenkins and J. F. Traub. Algorithm 419: zeros of a complex polynomial [c2]. *Commun. ACM*, 15(2):97–99, 1972.
- [36] D. M. Johnson, A. L. Dulmage, and N. S. Mendelsohn. Connectivity and reducibility of graphs. *Can. J. Math.*, 14:529–539, 1962.
- [37] Thomas Johnsson. Lambda lifting: transforming programs to recursive equations. In *Proceedings of the IFIP Conference on Functional Programming Languages and Computer Architecture*, 1985.
- [38] David C. Keenan. To dissect a mockingbird: A graphical notation for the lambda calculus with animated reduction. <http://users.bigpond.net.au/d.keenan/Lambda/>, May 2001.
- [39] B.W. Kernighan and D.M. Ritchie. *The C programming language*. Prentice-Hall, 1988.
- [40] Stephen Cole Kleene. *Mathematical Logic*. Dover, 2002.

- [41] Konrad Knopp. *Theory of Functions, Parts I and II*. Dover, 1996.
- [42] P. Kunkel and V. Mehrmann. *Differential-algebraic equations: analysis and numerical solution*. European Mathematical Society, 2006.
- [43] Casimir Kuratowski. Sur la notion de l'ordre dans la theorie des ensembles. *Fundamenta Mathematicae*, 2:161–171, 1921.
- [44] Chi-Chung Lam, Daniel Cociorva, Gerald Baumgartner, and P. Sadayappan. Memory-optimal evaluation of expression trees involving large objects. In *High Performance Computing HiPC'99*, volume 1745/2004 of *Lecture Notes in Computer Science*, pages 103–110. Springer Berlin/Heidelberg, 1 June 2004.
- [45] Suresh K. Lodha and Ron Goldman. A unified approach to evaluation algorithms for multivariate polynomials. *Mathematics of Computation*, 66(220):1521–1553, October 1997.
- [46] B. Malone and M. Paypay. ModelCenter: an integration environment for simulation based design. <http://www.phoenix-int.com/library/papers.php>.
- [47] The MathWorks. Simulink 7 simulation and model-based design. Technical report, The MathWorks, September 2007.
- [48] J. McCarthy. Recursive functions symbolic expressions and their computation by machine, Part I. *Communications of the ACM*, 3(4):184–195, 1960.
- [49] A. McInnes. Integrated Concurrent Design: making the conceptual design process faster, better, and cheaper, 2003.
- [50] Elliott Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall, 4th edition, 1997.
- [51] Joel Moses. The function of FUNCTION in LISP, or why the FUNARG problem should be called the environment problem. Technical report, Massachusetts Institute of Technology, June 1970.
- [52] K.L.G. Parkin, J.C. Sercel, and M.J. Liu. ICEMaker: an Excel-based environment for collaborative design. In *IEEE Aerospace Conference Proceedings*, volume 8, pages 3669–3679, 2003.
- [53] J. M. Peña and Thomas Sauer. On the multivariate Horner scheme. *SIAM J Numer Anal*, 37(4):1186–1197, 11 April 2000.
- [54] Alex Pothén. *Sparse null bases and marriage theorems*. PhD thesis, Cornell University, Ithaca, NY, 1984.
- [55] S. Presley and J. Neff. Implementing a concurrent design process — the human element is the most powerful part of the system. In *IEEE Aerospace Conference Proceedings*, 2000.

- [56] G. Raeder. A survey of current graphical programming techniques. *Computer*, 18(8):11–25, 1985.
- [57] A.M. Ross, D.E. Hastings, J.M. Warmkessel, and N.P. Diller. Multi-Attribute Tradespace Exploration as front end for effective space system design. *Journal of Spacecraft and Rockets*, 41(1):20–28, 2004.
- [58] Walter Rudin. *Real and complex analysis*. McGraw-Hill, 1987.
- [59] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial Mathematics, 2003.
- [60] G. Sanders. The sky’s the limit for CIEL. *BSS World*, 3(13), 2002.
- [61] Thoralf A. Skolem. Einige Bemerkungen zur axiomatischen Begründung der Mengenlehre. In *Den femte Skandinaviska Matematikerkongressen*, pages 217–232, 1922.
- [62] J.L. Smith. Concurrent Engineering in the JPL Project Design Center. Technical Report 98AMTC-83, Society of Automotive Engineers, 1998.
- [63] G.L. Steele. *Common LISP: The Language*. Digital Press, 1990.
- [64] Patrick Suppes. *Axiomatic Set Theory*. Dover, 1972.
- [65] Alfred Tarski. Sur quelques théorèmes qui équivalent à l’axiome du choix. *Fundamenta Mathematicae*, 5:147–154, 1924.
- [66] J. Tromp. Binary Lambda Calculus and Combinatory Logic, 2004.
- [67] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42 of 2, pages 230–265, 1936.
- [68] David A. Turner. SASL language manual. Technical report, Department of Computational Science, St. Andrews University, 1976.
- [69] David A. Turner. Another algorithm for bracket abstraction. *Journal of Symbolic Logic*, 44(2):267–270, 1979.
- [70] David A. Turner. A new implementation technique for applicative languages. *Software Practice and Experience*, 9(1):31–49, 1979.
- [71] David A. Turner. An overview of Miranda. *SIGPLAN Notices*, 21(12):158–166, 1986.
- [72] David A. Turner. Church’s Thesis and functional programming. In A. Olszewski, editor, *Church’s Thesis after 70 years*, chapter Church’s Thesis and functional programming. Logos Verlag, 2006.

- [73] J. V. Uspensky and M. A. Heaslet. *Elementary Number Theory*. McGraw-Hill, 1939.
- [74] C. P. Wadsworth. *The semantics and pragmatics of the lambda calculus*. PhD thesis, Oxford University, 1971.
- [75] S.D. Wall, D.B. Smith, and L.J. Koenig. Team structures and processes in the design of space missions. *IEEE Aerospace Conference Proceedings*, 2:35–42, 1999.
- [76] V. Weispfenning. Comprehensive Grobner bases. *Journal of Symbolic Computation*, 14(1):1–29, 1992.
- [77] Norbert Wiener. A simplification of the logic of relations. *Proceedings of the Cambridge Philosophical Society*, 17:387–390, 1914.
- [78] Ernst Zermelo. Untersuchungen über die Grundlagen der Mengenlehre I. *Mathematische Annalen*, 65:261–281, 1908.